pima

September 7, 2024

[4]:
```python
import numpy as np # linear algebra
import pandas as pd
pd.set_option("display.max_columns", None)
data=pd.read_csv('diabetes.csv')
data.head()
```

[4]:
```
   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
```

[5]:
```python
data.shape
```

[5]: (768, 9)

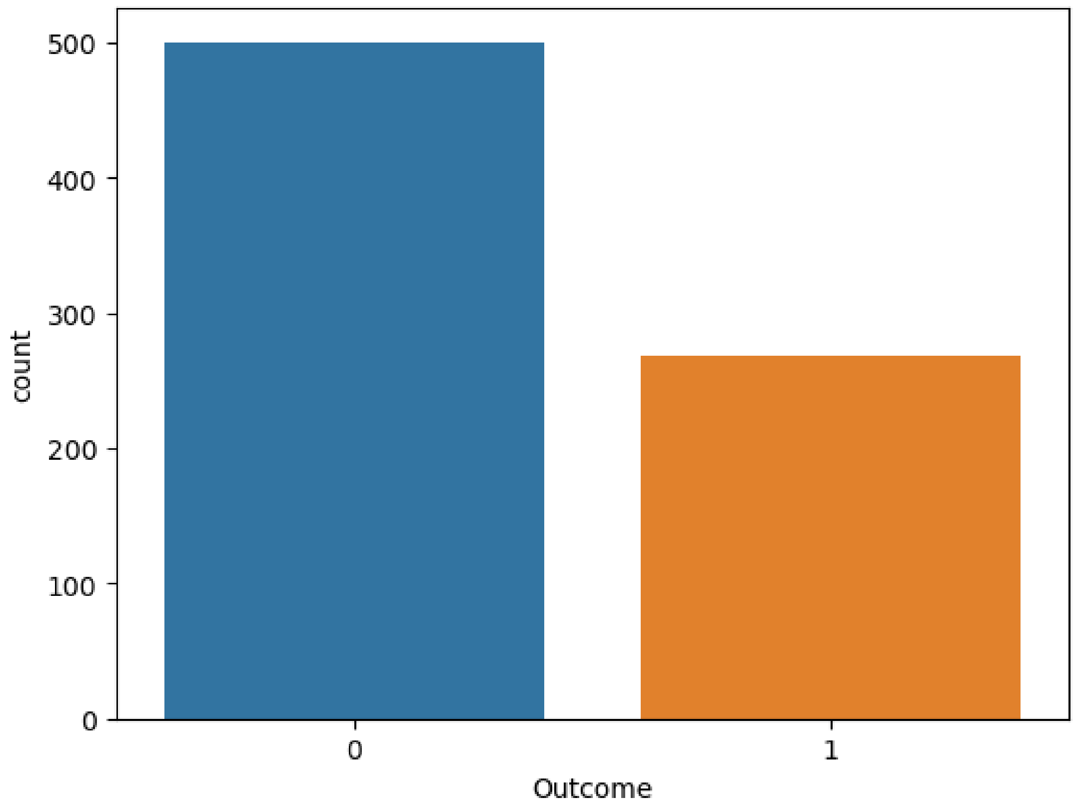
[6]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

[8]:
```python
y=data["Outcome"]
data.columns
```

[8]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

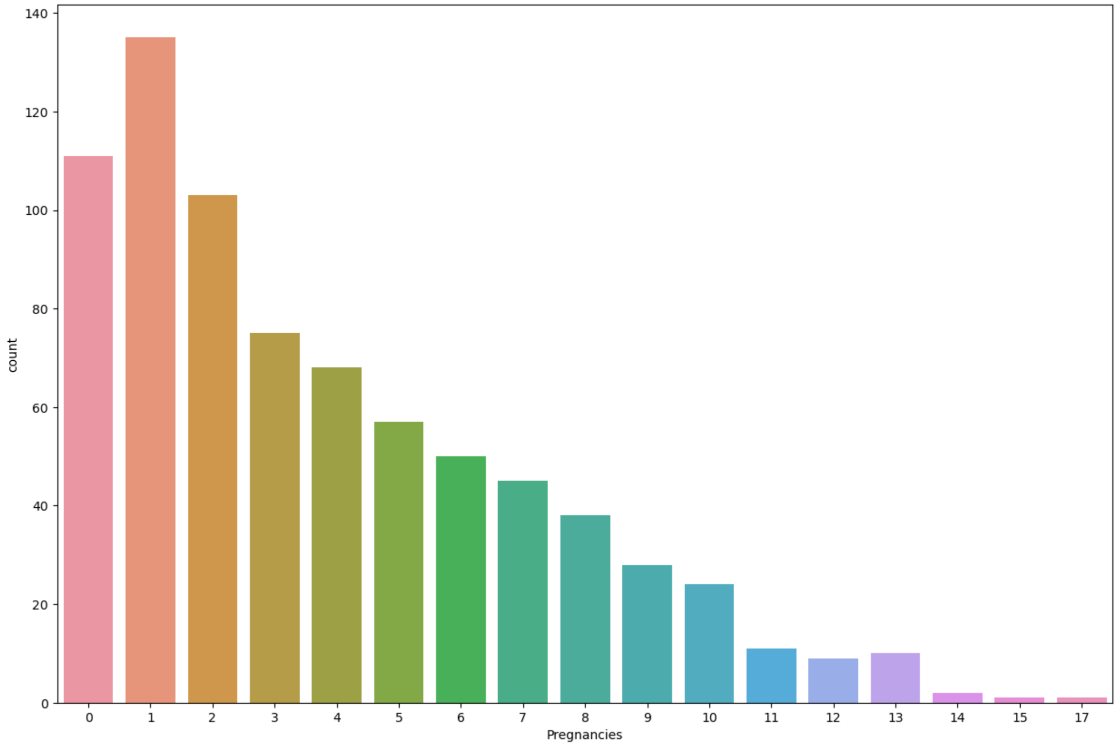
[10]:
```python
sns.countplot(data = data, x = data['Outcome']) # To check whether the data is␣
 ↪balanced or unbalanced
```

[10]: <Axes: xlabel='Outcome', ylabel='count'>



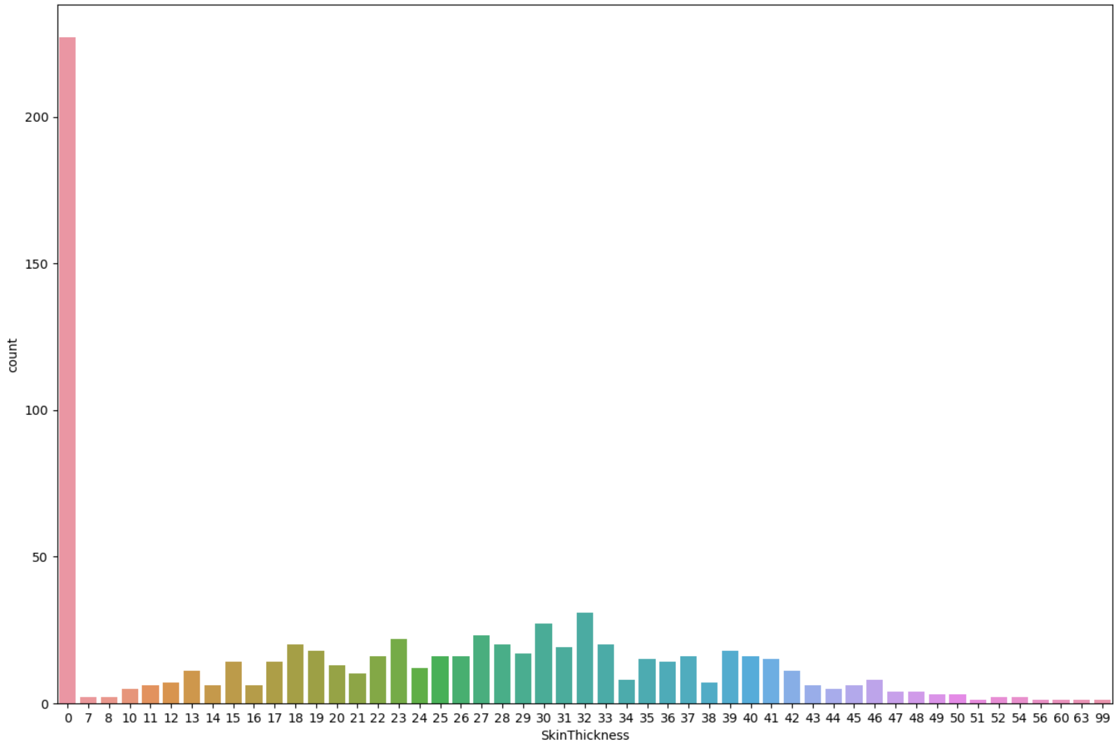
[12]:
```
fig = plt.figure(figsize = (15,10))

sns.countplot(data = data, x = data['Pregnancies'])
```

[12]: <Axes: xlabel='Pregnancies', ylabel='count'>



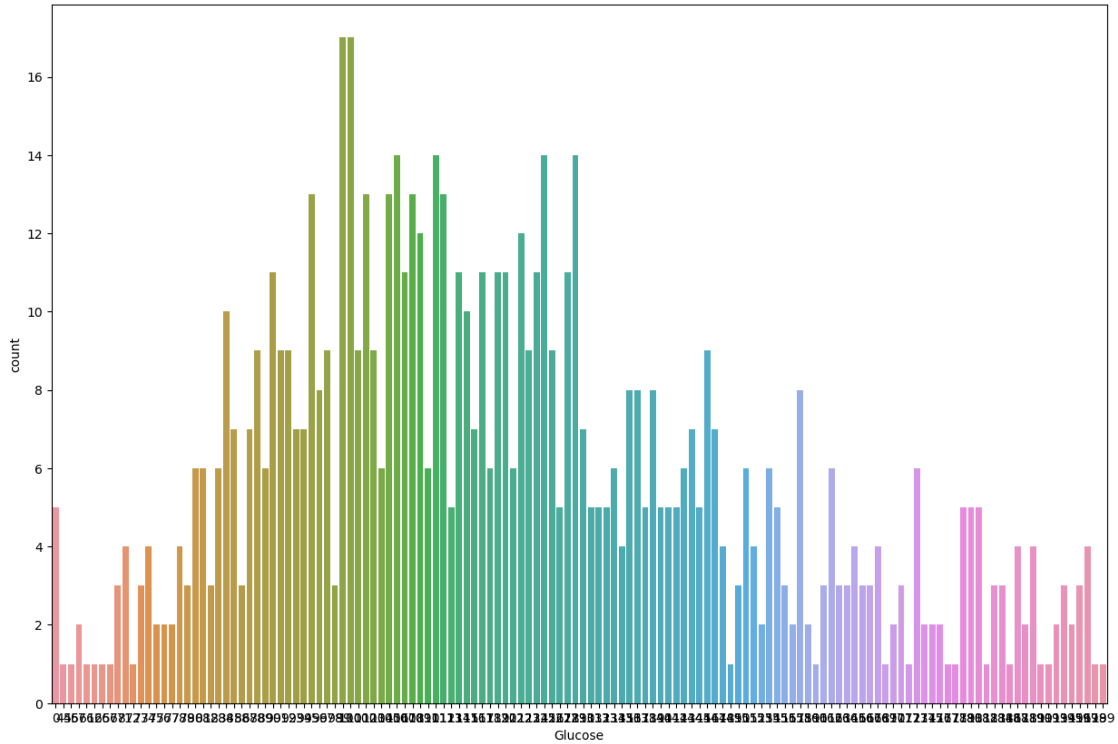
[13]:
```
fig = plt.figure(figsize = (15,10))
sns.countplot(data = data, x = data['SkinThickness'])
```

[13]: <Axes: xlabel='SkinThickness', ylabel='count'>



[15]:
```
fig = plt.figure(figsize = (15,10))
sns.countplot(data = data, x = data['Glucose'])
```

[15]: <Axes: xlabel='Glucose', ylabel='count'>



[16]:
```
data.describe()
```

[16]:
```
       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
```

[17]:
```
data.corr()
```

[17]:

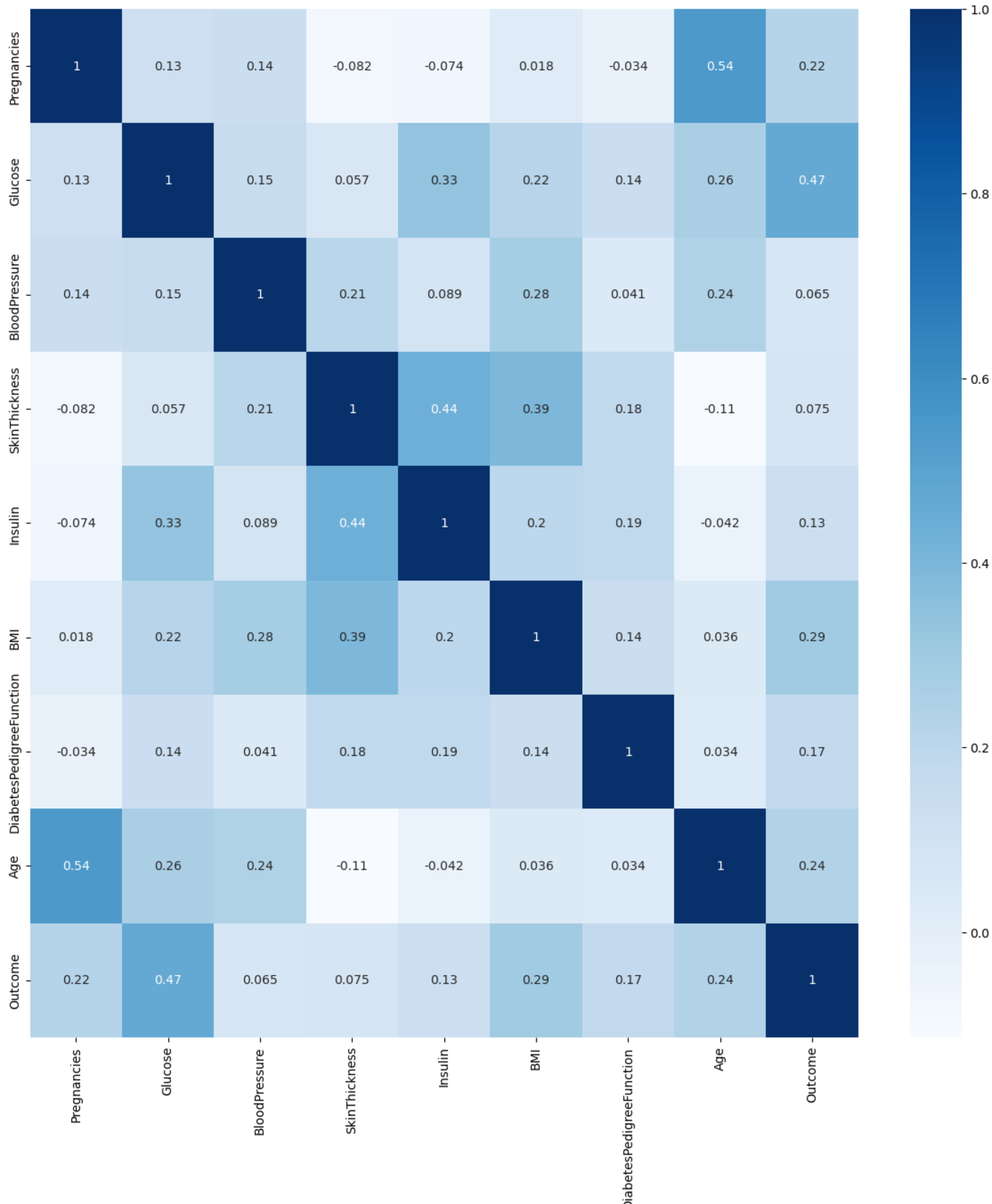
	Pregnancies	Glucose	BloodPressure	SkinThickness	\
Pregnancies	1.000000	0.129459	0.141282	-0.081672	
Glucose	0.129459	1.000000	0.152590	0.057328	
BloodPressure	0.141282	0.152590	1.000000	0.207371	
SkinThickness	-0.081672	0.057328	0.207371	1.000000	
Insulin	-0.073535	0.331357	0.088933	0.436783	
BMI	0.017683	0.221071	0.281805	0.392573	
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	
Age	0.544341	0.263514	0.239528	-0.113970	
Outcome	0.221898	0.466581	0.065068	0.074752	

	Insulin	BMI	DiabetesPedigreeFunction	\
Pregnancies	-0.073535	0.017683	-0.033523	
Glucose	0.331357	0.221071	0.137337	
BloodPressure	0.088933	0.281805	0.041265	
SkinThickness	0.436783	0.392573	0.183928	
Insulin	1.000000	0.197859	0.185071	
BMI	0.197859	1.000000	0.140647	
DiabetesPedigreeFunction	0.185071	0.140647	1.000000	
Age	-0.042163	0.036242	0.033561	
Outcome	0.130548	0.292695	0.173844	

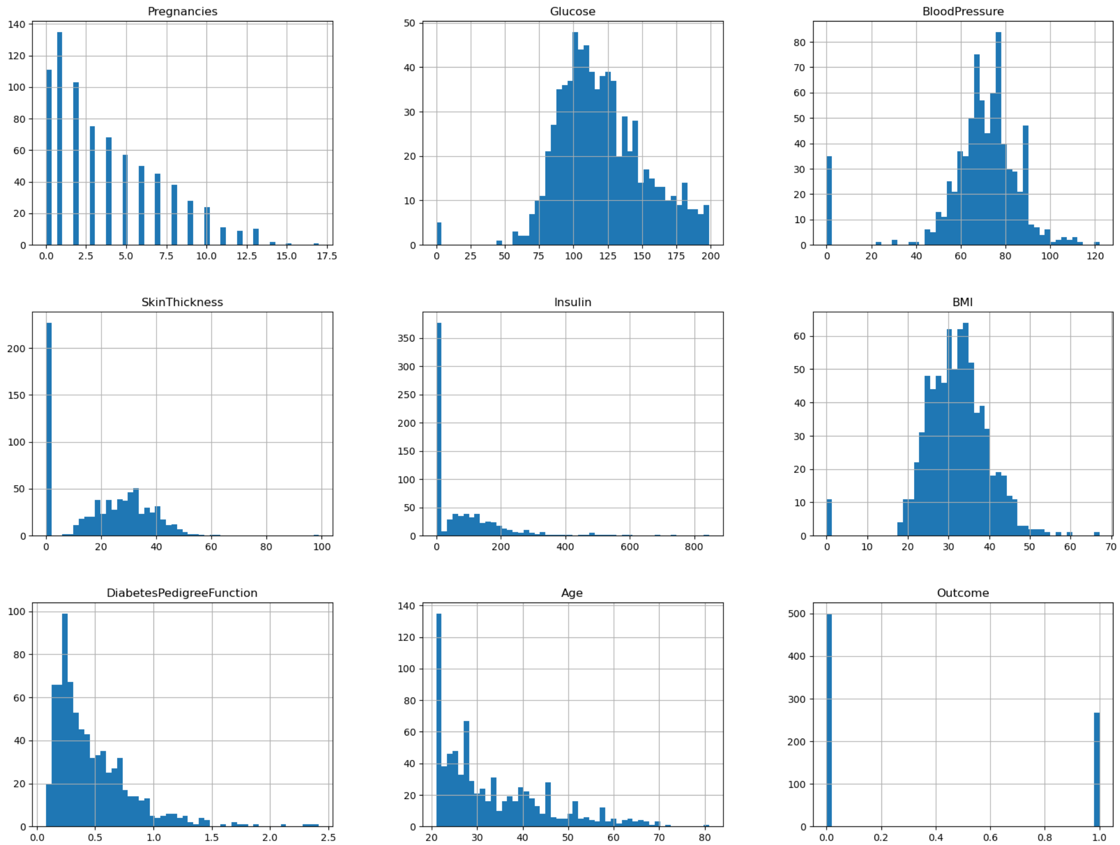
	Age	Outcome
Pregnancies	0.544341	0.221898
Glucose	0.263514	0.466581
BloodPressure	0.239528	0.065068
SkinThickness	-0.113970	0.074752
Insulin	-0.042163	0.130548
BMI	0.036242	0.292695
DiabetesPedigreeFunction	0.033561	0.173844
Age	1.000000	0.238356
Outcome	0.238356	1.000000

[20]:

```python
import seaborn as sb
corr = data.corr()
fig, ax = plt.subplots(figsize=(15, 15))
sb.heatmap(corr, cmap="Blues", annot=True, ax= ax)
plt.show() #Heatmap of the correlation plot
```



[21]:
```
data.hist(bins=50,figsize=(20,15))
plt.show()
```



[22]:
```
data=data.drop(["Outcome"],axis=1)
```

[23]:
```
data.head()
```

[23]:
```
   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
0                     0.627   50
1                     0.351   31
2                     0.672   32
3                     0.167   21
4                     2.288   33
```

[24]:
```
data.dtypes
```

[24]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
dtype: object
```

[25]:
```
categorical_cols = ['Pregnancies']
```

[26]:
```
data[categorical_cols] = data[categorical_cols].astype('category')
data.shape
```

[26]:
```
(768, 8)
```

[29]:
```
from sklearn.metrics import f1_score, accuracy_score, roc_auc_score,␣
 ↪confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import cross_val_score, GridSearchCV,␣
 ↪train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, MinMaxScaler
xtrain,xval,ytrain,yval=train_test_split(data,y,random_state=0,test_size=0.
 ↪2,shuffle=True,stratify=y)
```

[30]:
```
xtrain.shape,xval.shape,ytrain.shape,ytrain.shape
```

[30]:
```
((614, 8), (154, 8), (614,), (614,))
```

[37]:
```
from sklearn.ensemble import RandomForestClassifier   # Classification using␣
 ↪Randome forest classifier
RFclf = RandomForestClassifier()
model = RFclf.fit(xtrain, ytrain)
```

[38]:
```
score=cross_val_score(model,xval,yval,cv=5)
```

[39]:
```
model.score(xval,yval)
```

[39]:
```
0.7987012987012987
```

[40]:
```
ypredict=model.predict(xval)
cm=confusion_matrix(yval,ypredict)
```

[41]:
```
import sklearn
pd.DataFrame(sklearn.metrics.classification_report(yval, ypredict,␣
 ↪output_dict=True)).T
```

[41]:

	precision	recall	f1-score	support
0	0.816514	0.890000	0.851675	100.000000
1	0.755556	0.629630	0.686869	54.000000
accuracy	0.798701	0.798701	0.798701	0.798701
macro avg	0.786035	0.759815	0.769272	154.000000
weighted avg	0.795139	0.798701	0.793886	154.000000

[42]:
```
model.feature_importances_
```

[42]:
```
array([0.08486638, 0.24726198, 0.09153658, 0.06728744, 0.07230195,
       0.15698264, 0.12186403, 0.15789901])
```

[43]:
```
model.feature_names_in_
```

[43]:
```
array(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
       'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'], dtype=object)
```

[76]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

[77]:
```
scaler = MinMaxScaler()
x = xtrain.values
trained_scaler = scaler.fit(x)
```

[78]:
```
x_train_scaled = scaler.transform(x)
```

[79]:
```
x_val_scaled = scaler.transform(xval.values)
```

[80]:
```
from sklearn.linear_model import LogisticRegression
LRclf = LogisticRegression()
model = LRclf.fit(x_train_scaled, ytrain)
```

[81]:
```
y_predicted = model.predict(x_val_scaled)
```

[82]:
```
import sklearn
pd.DataFrame(sklearn.metrics.classification_report(yval, y_predicted,␣
 ↪output_dict=True)).T
```

[82]:

	precision	recall	f1-score	support
0	0.771186	0.910000	0.834862	100.000000
1	0.750000	0.500000	0.600000	54.000000
accuracy	0.766234	0.766234	0.766234	0.766234
macro avg	0.760593	0.705000	0.717431	154.000000
weighted avg	0.763757	0.766234	0.752508	154.000000

[83]:
```
param_grid = [
    {'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'C' : np.logspace(-4, 4, 20),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga'],
```

```
    'max_iter' : [100, 1000,2500, 5000]
    }
]
```

[84]:
```
clf = GridSearchCV(LRclf, param_grid = param_grid, cv = 3, verbose=True,␣
 ↪n_jobs=-1)
```

[101]:
```
#best_clf = clf.fit(x_train_scaled, ytrain)
```

[86]:
```
best_clf.best_estimator_
```

[86]:
```
LogisticRegression(C=11.288378916846883, penalty='l1', solver='saga')
```

[87]:
```
y_predicted=best_clf.best_estimator_.predict(x_val_scaled)
```

[88]:
```
pd.DataFrame(sklearn.metrics.classification_report(yval, y_predicted,␣
 ↪output_dict=True)).T
```

[88]:

	precision	recall	f1-score	support
0	0.794643	0.890000	0.839623	100.000000
1	0.738095	0.574074	0.645833	54.000000
accuracy	0.779221	0.779221	0.779221	0.779221
macro avg	0.766369	0.732037	0.742728	154.000000
weighted avg	0.774814	0.779221	0.771671	154.000000

[89]:
```
import tensorflow  # Loading required libraries for neural network
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
```

[90]:
```
ann = Sequential()

ann.add(Dense(20,activation='relu',input_dim=8))
#model.add(BatchNormalization())
ann.add(Dense(50,activation='relu'))
ann.add(Dense(50,activation='relu'))
#ann.add(Dense(200,activation='relu'))
#model.add(BatchNormalization())
#ann.add(Dense(200,activation='relu'))
ann.add(Dense(1,activation='sigmoid'))
```

[91]:
```
ann.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

[92]:
```
callback = EarlyStopping(
    monitor="val_loss", # which quantity to be measured to check
```

```
    min_delta=0.0001, #minimum change in monitored quantity that would qualify␣
↪as improvement.
    patience=20,  #number of epochs passed with no improvement when process␣
↪will stop.
    verbose=1,  #output print or not.
    mode="auto", # min, max , auto
    baseline=None, # baseline value of monitored quantity.
    restore_best_weights=True,
)
```

[93]:

```
history1 = ann.fit(x_train_scaled, ytrain,epochs=20,validation_split=0.
↪2,verbose=1,callbacks=callback) # NN Training
```

```
Epoch 1/20
16/16 [==============================] - 0s 4ms/step - loss: 0.6742 - accuracy:
0.6517 - val_loss: 0.6629 - val_accuracy: 0.6504
Epoch 2/20
16/16 [==============================] - 0s 1ms/step - loss: 0.6630 - accuracy:
0.6517 - val_loss: 0.6558 - val_accuracy: 0.6504
Epoch 3/20
16/16 [==============================] - 0s 1ms/step - loss: 0.6561 - accuracy:
0.6517 - val_loss: 0.6483 - val_accuracy: 0.6504
Epoch 4/20
16/16 [==============================] - 0s 1ms/step - loss: 0.6470 - accuracy:
0.6517 - val_loss: 0.6404 - val_accuracy: 0.6504
Epoch 5/20
16/16 [==============================] - 0s 1ms/step - loss: 0.6402 - accuracy:
0.6517 - val_loss: 0.6288 - val_accuracy: 0.6504
Epoch 6/20
16/16 [==============================] - 0s 1ms/step - loss: 0.6313 - accuracy:
0.6538 - val_loss: 0.6164 - val_accuracy: 0.6504
Epoch 7/20
16/16 [==============================] - 0s 1ms/step - loss: 0.6110 - accuracy:
0.6701 - val_loss: 0.6115 - val_accuracy: 0.7398
Epoch 8/20
16/16 [==============================] - 0s 1ms/step - loss: 0.5916 - accuracy:
0.7088 - val_loss: 0.5852 - val_accuracy: 0.6911
Epoch 9/20
16/16 [==============================] - 0s 1ms/step - loss: 0.5787 - accuracy:
0.7088 - val_loss: 0.5751 - val_accuracy: 0.7398
Epoch 10/20
16/16 [==============================] - 0s 1ms/step - loss: 0.5559 - accuracy:
0.7047 - val_loss: 0.5553 - val_accuracy: 0.7561
Epoch 11/20
16/16 [==============================] - 0s 1ms/step - loss: 0.5383 - accuracy:
0.7230 - val_loss: 0.5511 - val_accuracy: 0.7724
Epoch 12/20
16/16 [==============================] - 0s 1ms/step - loss: 0.5213 - accuracy:
```

```
0.7475 - val_loss: 0.5415 - val_accuracy: 0.7561
Epoch 13/20
16/16 [==============================] - 0s 1ms/step - loss: 0.5061 - accuracy:
0.7597 - val_loss: 0.5366 - val_accuracy: 0.7480
Epoch 14/20
16/16 [==============================] - 0s 1ms/step - loss: 0.4995 - accuracy:
0.7617 - val_loss: 0.5730 - val_accuracy: 0.7154
Epoch 15/20
16/16 [==============================] - 0s 1ms/step - loss: 0.4946 - accuracy:
0.7658 - val_loss: 0.5594 - val_accuracy: 0.7317
Epoch 16/20
16/16 [==============================] - 0s 1ms/step - loss: 0.4892 - accuracy:
0.7617 - val_loss: 0.5805 - val_accuracy: 0.7154
Epoch 17/20
16/16 [==============================] - 0s 1ms/step - loss: 0.4796 - accuracy:
0.7719 - val_loss: 0.5441 - val_accuracy: 0.7480
Epoch 18/20
16/16 [==============================] - 0s 1ms/step - loss: 0.4687 - accuracy:
0.7821 - val_loss: 0.5646 - val_accuracy: 0.7317
Epoch 19/20
16/16 [==============================] - 0s 1ms/step - loss: 0.4692 - accuracy:
0.7780 - val_loss: 0.5658 - val_accuracy: 0.7398
Epoch 20/20
16/16 [==============================] - 0s 1ms/step - loss: 0.4644 - accuracy:
0.7760 - val_loss: 0.5589 - val_accuracy: 0.7398
```

[95]:

```
from sklearn.metrics import
 ↪f1_score,accuracy_score,roc_auc_score,confusion_matrix,ConfusionMatrixDisplay
ypredict=ann.predict(x_val_scaled)

ann.evaluate(x_val_scaled, yval)
```

```
5/5 [==============================] - 0s 1ms/step
5/5 [==============================] - 0s 778us/step - loss: 0.4347 - accuracy:
0.8182
```

[95]: [0.4347224533557892, 0.8181818127632141]

[96]:

```
rounded = [int(round(x[0])) for x in ypredict]
print(rounded)
```

```
[0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1,
1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0]
```

[97]:

```
y_pred=pd.DataFrame(rounded,columns=['ypred'])
```

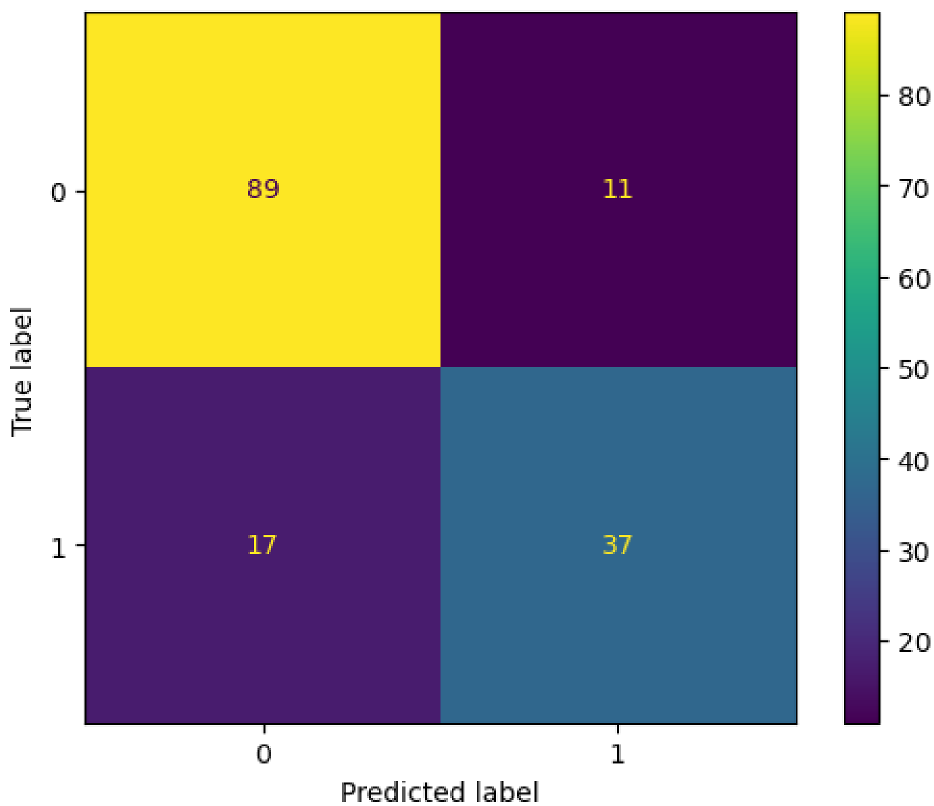
[98]:
```python
y_pred.ypred.value_counts()
```

[98]:
```
ypred
0    106
1     48
Name: count, dtype: int64
```

[99]:
```python
pd.DataFrame(sklearn.metrics.classification_report(yval, rounded,␣
↪output_dict=True)).T
```

[99]:
```
              precision    recall  f1-score     support
0              0.839623  0.890000  0.864078  100.000000
1              0.770833  0.685185  0.725490   54.000000
accuracy       0.818182  0.818182  0.818182    0.818182
macro avg      0.805228  0.787593  0.794784  154.000000
weighted avg   0.815502  0.818182  0.815482  154.000000
```

[100]:
```python
y_pred= np.round(ypredict).tolist()
cm=confusion_matrix(yval,y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
cm_display.plot()
plt.show()
```



```
[ ]:
```