```
#PCOS or polycystic ovary syndrome :
#PCOS is a hormonal imbalance that affects ovulation. This can cause irregular periods, excess androgen, and cysts in the ovaries. It's a common condi

# PCOS Dataset Source : https://www.kaggle.com/datasets/cm037divya/pcos-dataset

"""Importing libraries"""
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

"""Reading the dataset using pandas"""

dataset = pd.read_csv("/content/PCOS/PCOS_extended_dataset.csv")

"""Analysis of Dataset"""

dataset.head(10)

dataset.tail(10)

"""Getting the dimension of dataset"""

rows = dataset.shape[0]
columns = dataset.shape[1]
print("There are",rows,"rows and",columns,"Columns in dataset")

"""Getting metadata about data (Datatype of column and count of non-null values)"""

dataset.info()

"""Getting summary statistics of dataset"""

dataset.describe()

"""Printing the columns name of dataset"""

print(dataset.columns)

"""Cleaning"""

dataset_1 = dataset.drop(['Sl. No', 'Patient File No.','Weight (Kg)','Height(Cm) ','Hip(inch)', 'Waist(inch)','Marraige Status (Yrs)'], axis=1)

dataset_1.isna().sum()

"""Updating column datatypes from object to float"""

for col in dataset_1:
    if dataset_1[col].dtypes == object:
      dataset_1[col] = pd.to_numeric(dataset_1[col], errors = "coerce") # The errors="coerce" parameter instructs pandas to convert any values that c

"""Select all columns except the first ('PCOS(Y/N)') as it represents the target variable"""

columns = dataset_1.columns[1:]

"""Create a grid of boxplots to visually inspect the distribution of each feature (excluding target) and identify potential outliers."""

plt.figure(figsize = (35,35))
for i, col in enumerate(columns):
    plt.subplot(9, 4, i+1)
    sns.boxplot(x = dataset_1[col])
plt.show()

"""Calculate pairwise correlation coefficients"""

correlation = dataset_1.corr()
correlation

"""Generate a heatmap of the correlation matrix with annotations"""

plt.figure(figsize = (40,40))
sns.heatmap(correlation, annot=True)   # it's a 37x37 correlation matrix,

"""Removing the outliers from the dataset"""

outliers_columns = ['Pulse rate(bpm) ','FSH(mIU/mL)','LH(mIU/mL)','FSH/LH','TSH (mIU/L)','Vit D3 (ng/mL)','BP _Systolic (mmHg)','BP _Diastolic (mmHg)
lower_range = [60,0,0.05,0.39,0.4,5,80,50,1]
upper_range = [100,4000,30,17,7,90,145,105,19]

outliers_df = pd.DataFrame({'Columns': outliers_columns, 'Lower Range': lower_range, 'Upper Range': upper_range})
print(outliers_df.to_string())

def get_outliers_index(data, lower, upper):
  store=[]
  for i in range(len(data)):
    if (data[i]>upper or data[i]<lower):
      store.append(i)
  return store

outlier_index=[]
for i in range(len(outliers_columns)):
  data=outliers_columns[i]
  lower=lower_range[i]
  upper=upper_range[i]
  indexes=get_outliers_index(dataset_1[data], lower, upper)
  for i in indexes:
```

```
  for j in indexes:
    if j not in outlier_index:
      outlier_index.append(j)

print("Total number of Outliers found : ", len(outlier_index))

Cleaned_Dataset = dataset_1.drop(outlier_index)

"""Cleaned Dataset"""

rows = Cleaned_Dataset.shape[0]
columns = Cleaned_Dataset.shape[1]
print("There are", rows, "rows and", columns, "Columns in Cleaned Dataset")

plt.figure(figsize = (40,40))
sns.heatmap(Cleaned_Dataset.corr(), annot=True)   # it's a 37x37 correlation matrix,

"""Splitting Dataset"""

Dataset = Cleaned_Dataset.dropna()
X=Dataset.iloc[:,1:].values # Independent variables
Y=Dataset.iloc[:,0].values # Target variable

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)

#Implementing ANN

ann_model = keras.Sequential([
    keras.layers.Dense(16, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
])
ann_model.build(input_shape=(1,36))
ann_model.summary()

ann_model.compile(optimizer="adam",loss="binary_crossentropy",metrics=['accuracy'])
ann_model.fit(X_train,Y_train,epochs=70,batch_size=32)
Model_Name=[]
Accuracy_Model=[]

loss, accuracy = ann_model.evaluate(X_test, Y_test)
print('Loss:', loss)
print('Accuracy:', accuracy)
predicted=ann_model.predict(X_test)
predicted = [1 if x > 0.5 else 0 for x in predicted.flatten()]
cm = confusion_matrix(Y_test,predicted)
sns.heatmap(cm,
   annot=True,
   fmt='g',
   xticklabels=['PCOS +','PCOS -'],
   yticklabels=['PCOS +','PCOS -'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()


Model_Name.append('ANN')
Accuracy_Model.append(accuracy)

"""Implementing SVM"""

kernel_Name=[ 'linear', 'sigmoid', 'rbf', 'poly']
Differ_Kernel_Accuracy=[]

for Kernel in kernel_Name:
  classifier=SVC(kernel=Kernel)
  classifier.fit(X_train,Y_train)
  predicted=classifier.predict(X_test)
  Differ_Kernel_Accuracy.append(accuracy_score(Y_test,predicted))

SVM_Accuracy = pd.DataFrame({'Kernel USed': kernel_Name, 'Accuracy': Differ_Kernel_Accuracy})
print(SVM_Accuracy.to_string())

print("SVM gives Max accuracy of ",max(Differ_Kernel_Accuracy)," with kernel ",kernel_Name[np.array(Differ_Kernel_Accuracy).argmax()],"\n")
SVM_Model=SVC(kernel=kernel_Name[np.array(Differ_Kernel_Accuracy).argmax()])
SVM_Model.fit(X_train, Y_train)
predicted=SVM_Model.predict(X_test)
cm = confusion_matrix(Y_test, predicted)
sns.heatmap(cm,
   annot=True,
   fmt='g',
   xticklabels=['PCOS +','PCOS -'],
   yticklabels=['PCOS +','PCOS -'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()


Model_Name.append('SVM')
Accuracy_Model.append(max(Differ_Kernel_Accuracy))

"""Implementing Naive Bayes Classifier"""

gnb_model = GaussianNB()
gnb_model.fit(X_train, Y_train)

y_pred = gnb_model.predict(X_test)

cm = confusion_matrix(Y_test, y_pred)

sns.heatmap(cm,
   annot = True,
   fmt='g'
```

```
   fmt='g',
   xticklabels=['PCOS +','PCOS -'],
   yticklabels=['PCOS +','PCOS -'])
plt.ylabel('Prediction', fontsize=13)
plt.xlabel('Actual', fontsize=13)
plt.title('Confusion Matrix', fontsize=17)
plt.show()

accuracy = accuracy_score(y_pred, Y_test)
print('Accuracy : ', accuracy)

Model_Name.append('Naive Bayes')
Accuracy_Model.append(accuracy)

"""Implementing the Decision Tree Model"""

DT_Model = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
DT_Model.fit(X_train, Y_train)

y_pred = DT_Model.predict(X_test)

cm = confusion_matrix(Y_test, y_pred)

sns.heatmap(cm,
   annot = True,
   fmt='g',
   xticklabels=['PCOS +','PCOS -'],
   yticklabels=['PCOS +','PCOS -'])
plt.ylabel('Prediction', fontsize=13)
plt.xlabel('Actual', fontsize=13)
plt.title('Confusion Matrix', fontsize=17)
plt.show()

accuracy = accuracy_score(y_pred, Y_test)
print('Accuracy : ', accuracy)

Model_Name.append('Decision Tree')
Accuracy_Model.append(accuracy)

"""Implementing Random Forest Model"""

RF_model = RandomForestClassifier(n_estimators = 10, criterion = "entropy")
RF_model.fit(X_train, Y_train)

y_pred = RF_model.predict(X_test)

cm = confusion_matrix(Y_test, y_pred)

sns.heatmap(cm,
   annot = True,
   fmt='g',
   xticklabels=['PCOS +','PCOS -'],
   yticklabels=['PCOS +','PCOS -'])
plt.ylabel('Prediction', fontsize=13)
plt.xlabel('Actual', fontsize=13)
plt.title('Confusion Matrix', fontsize=17)
plt.show()

accuracy = accuracy_score(y_pred, Y_test)
print('Accuracy : ', accuracy)


Model_Name.append('Random Forest')
Accuracy_Model.append(accuracy)

Models = pd.DataFrame({'Model Name': Model_Name, 'Accuracy': Accuracy_Model})
print(Models.to_string())
```