```
# -*- coding: utf-8 -*-
"""Untitled6.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/19eMihED1qJQu3pq5naGoIjLGMkGGs4BQ
"""

# for whole program
import numpy as py
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

# for knn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.model_selection import GridSearchCV

# for decision tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.model_selection import GridSearchCV

# reading dataset
df = pd.read_csv('diabetes.csv')

# print first 5 rows
df.head()

# print number of items and columns
df.shape

# find some statistical values about the dataset
df.describe()

# dropping duplicate values
df=df.drop_duplicates()

# checking for null/0 values
```

```
df.isnull().sum()

# replace 0 values with median of that particular column
df['Glucose']=df['Glucose'].replace(0,df['Glucose'].mean())
df['BloodPressure']=df['BloodPressure'].replace(0,df['BloodPressure'].mean())
df['SkinThickness']=df['SkinThickness'].replace(0,df['SkinThickness'].median())
df['Insulin']=df['Insulin'].replace(0,df['Insulin'].median())
df['BMI']=df['BMI'].replace(0,df['BMI'].median())

# displaying the correlation coefficient
corrmat=df.corr()
sns.heatmap(corrmat, annot=True)

# count the number of outcomes
sns.countplot(x = 'Outcome', data=df)

# histogram of the selected features
df.hist(bins=10,figsize=(10,10))
plt.show()

# plotting box plots to identify the outliers
plt.figure(figsize=(16,12))
sns.set_style(style='whitegrid')
plt.subplot(3,3,1)
sns.boxplot(x='Glucose',data=df)
plt.subplot(3,3,2)
sns.boxplot(x='BloodPressure',data=df)
plt.subplot(3,3,3)
sns.boxplot(x='Insulin',data=df)
plt.subplot(3,3,4)
sns.boxplot(x='BMI',data=df)
plt.subplot(3,3,5)
sns.boxplot(x='Age',data=df)
plt.subplot(3,3,6)
sns.boxplot(x='SkinThickness',data=df)
plt.subplot(3,3,7)
sns.boxplot(x='Pregnancies',data=df)
plt.subplot(3,3,8)
sns.boxplot(x='DiabetesPedigreeFunction',data=df)

# dropping the least correlated columns
df_selected=df.drop(['BloodPressure','Insulin','DiabetesPedigreeFunction'],axis='columns')

# transforming the values in columns to handle outliers in the dataset
from sklearn.preprocessing import QuantileTransformer
x=df_selected
quantile  = QuantileTransformer()
```

```
X = quantile.fit_transform(x)
df_new=quantile.transform(X)
df_new=pd.DataFrame(X)
df_new.columns =['Pregnancies', 'Glucose','SkinThickness','BMI','Age','Outcome']
df_new.head()

# new box plot that shows normal distribution now
plt.figure(figsize=(16,12))
sns.set_style(style='whitegrid')
plt.subplot(3,3,1)
sns.boxplot(x=df_new['Glucose'],data=df_new)
plt.subplot(3,3,2)
sns.boxplot(x=df_new['BMI'],data=df_new)
plt.subplot(3,3,3)
sns.boxplot(x=df_new['Pregnancies'],data=df_new)
plt.subplot(3,3,4)
sns.boxplot(x=df_new['Age'],data=df_new)
plt.subplot(3,3,5)
sns.boxplot(x=df_new['SkinThickness'],data=df_new)

# splitting the dataset
target_name='Outcome'
y= df_new[target_name]
X=df_new.drop(target_name,axis=1) #dropping target column from other columns in X

# splitting the dataset for training and testing purposes
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X,y,test_size=0.2,random_state=0)

# knn model
# hyperparameters are selected
knn= KNeighborsClassifier()
n_neighbors = list(range(15,25))
p=[1,2]
weights = ['uniform', 'distance']
metric = ['euclidean', 'manhattan', 'minkowski']

# they are converted into a dictionary form
hyperparameters = dict(n_neighbors=n_neighbors, p=p,weights=weights,metric=metric)

#actual model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=knn, param_grid=hyperparameters, n_jobs=-1, cv=cv,
scoring='f1',error_score=0)

# finding the best model from all folds
best_model = grid_search.fit(X_train,y_train)
```

```
# best values of the hyperparameters in the model
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])

# prediction on testing set
knn_pred = best_model.predict(X_test)

# print performance metrics along with confusion matrix chart
print("Classification Report is:\n",classification_report(y_test,knn_pred))
print("\n F1:\n",f1_score(y_test,knn_pred))
print("\n Precision score is:\n",precision_score(y_test,knn_pred))
print("\n Recall score is:\n",recall_score(y_test,knn_pred))
print("\n Confusion Matrix:\n")
sns.heatmap(confusion_matrix(y_test,knn_pred))

# decision tree model
dt = DecisionTreeClassifier(random_state=42)

# parameters grid to use for searching for best model
params = {
    'max_depth': [5, 10, 20,25],
    'min_samples_leaf': [10, 20, 50, 100,120],
    'criterion': ["gini", "entropy"]
}

# prepare a grid search to search for best model
grid_search = GridSearchCV(estimator=dt,
                           param_grid=params,
                           cv=4, n_jobs=-1, verbose=1, scoring = "accuracy")

# finding best model
best_model=grid_search.fit(X_train, y_train)

# testing the model on test set
dt_pred=best_model.predict(X_test)

# print performance metrics along with confusion matrix chart
print("Classification Report is:\n",classification_report(y_test,dt_pred))
print("\n F1:\n",f1_score(y_test,dt_pred))
print("\n Precision score is:\n",precision_score(y_test,dt_pred))
print("\n Recall score is:\n",recall_score(y_test,dt_pred))
print("\n Confusion Matrix:\n")
sns.heatmap(confusion_matrix(y_test,dt_pred))
```