```
# -*- coding: utf-8 -*-
"""Alaiba_Nawaz_Day4_W4.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1zI93rU0HmpreO1TkRCEISXOO-NsAIilV
"""

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings("ignore")

data = pd.read_csv("/content/chronic_kidney_disease.csv")

data.info()

#reading file with meaningful column name
data = pd.read_csv("/content/chronic_kidney_disease.csv" , decimal = '.',
header = 0,
                  names = ['id', 'age', 'blood_pressure',
'specific_gravity', 'albumin', 'sugar',
                           'red_blood_cells', 'pus_cell',
'pus_cell_clumps', 'bacteria',
                           'blood_glucose_random', 'blood_urea',
'serum_creatinine', 'sodium',
                           'potassium', 'haemoglobin',
'packed_cell_volume', 'white_cell_count',
                           'red_cell_count', 'hypertension',
'diabetes_mellitus',
                           'coronary_artery_disease', 'appetite',
'pedal_edema', 'anaemia',
                           'classification'])

data.info()
```

```
#looking for missing values
data.isna().sum()

data.head()

data["red_cell_count"].unique()

#converting red_cell_count to numeric as we can see \t?
data["red_cell_count"] = pd.to_numeric(data["red_cell_count"] , errors =
"coerce")

data["white_cell_count"].unique()

#converting white_cell_count to numeric as we can see \t? and \t8400
data["white_cell_count"] = pd.to_numeric(data["white_cell_count"] , errors
= "coerce")

data["packed_cell_volume"].unique()

#converting packed_cell_volume to numeric as we can see \t? and \t43'
data["packed_cell_volume"] = pd.to_numeric(data["packed_cell_volume"] ,
errors = "coerce")

#checking the categorical variables now
cat_var = []
num_var = []

for var in data.columns:
    if data[var].dtype == 'object':
        cat_var.append(var)
    elif data[var].dtype == 'float64':
        num_var.append(var)

for var in cat_var:
    print(f"{var} unique values: {data[var].unique()}")

#we can see that diabetes_mellitus , coronary_artery_disease and
classification have errors
#Fixing errors of 'diabetes_mellitus'
data.diabetes_mellitus = data.diabetes_mellitus.str.strip()
```

```
data['diabetes_mellitus'].replace({'/tno','/tyes'}, {'no','yes'}, inplace
= True)
#Fixing errors of 'coronary_artery_disease'
data['coronary_artery_disease'].replace('\tno', 'no', inplace = True)
# Fixing errors of 'classification'
data['classification'].replace('ckd\t', 'ckd', inplace = True)

#imputing missing values

# Function to impute missing values with mean for numerical variables
def impute_mean(variable):
    data[variable].fillna(data[variable].mean(), inplace=True)

# Function to impute missing values with mode for categorical variables
def impute_mode(variable):
    mode_val = data[variable].mode().iloc[0]
    data[variable].fillna(mode_val, inplace=True)

# Impute missing values for numerical variables using mean imputation
for var in num_var:
    impute_mean(var)

# Impute missing values for categorical variables using mode imputation
for var in cat_var:
    impute_mode(var)

data.isna().sum()

#visualising age
data["age"].describe()

# Plot the age distribution using a KDE plot
plt.figure(figsize=(8, 6))
sns.kdeplot(data=data, x="age", fill=True, legend=True)
plt.title('Figure 1: The Age Distribution', fontsize=17)
plt.xlabel('Age (years)', fontsize=15)
plt.ylabel('Density', fontsize=15)
plt.show()

#visualising blood pressure
```

```
data["blood_pressure"].describe()

plt.figure(figsize=(8, 6))
plt.hist(data["blood_pressure"], bins=11, edgecolor='black')
plt.title('Figure 2: Blood Pressure (diastolic) Distribution',
fontsize=17)
plt.xlabel('Blood Pressure (mmHg)', fontsize=15)
plt.ylabel('Number of People', fontsize=15)
plt.grid(True)
plt.show()

#visualising random blood sugar
data['blood_glucose_random'].describe()

plt.figure(figsize=(8, 6))
plt.hist(data["blood_glucose_random"], bins=60, edgecolor='black')
plt.title('Figure 3: Random Blood Glucose Test Distribution', fontsize=17)
plt.xlabel('Random Blood Glucose Levels (mg/dL)', fontsize=15)
plt.ylabel('Number of People', fontsize=15)
plt.grid(True)
plt.show()

#visualising sodium
data['sodium'].describe()

plt.figure(figsize=(8, 6))
plt.hist(data["sodium"], bins=60, edgecolor='black')
plt.title('Figure 4: Blood Sodium Level Distribution', fontsize=17)
plt.xlabel('Sodium Levels (mEQ/L)', fontsize=15)
plt.ylabel('Number of People', fontsize=15)
plt.grid(True)
plt.show()

#visualising potassium
data['potassium'].describe()

plt.figure(figsize=(8, 6))
plt.hist(data["potassium"], bins=60, edgecolor='black')
plt.title('Figure 5: Blood Potassium Level Distribution', fontsize=17)
plt.xlabel('Potassium Levels (mEQ/L)', fontsize=15)
```

```
plt.ylabel('Number of People', fontsize=15)
plt.grid(True)
plt.show()

#plotting heatmap to see correlation
plt.figure(figsize=(15,8));
plt.title("Correlation",color="green")
sns.heatmap(data.corr(),linewidth=1,annot=True)

# Transform categorical variables to numerical values
encoded_data=pd.get_dummies(data, drop_first='True')

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix,
classification_report,accuracy_score
y = encoded_data['classification_notckd']
x = encoded_data.drop('classification_notckd', axis =1)

#spliting data 80% for training and 20% for testing
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_st
ate=222)

#applying logistic
model=LogisticRegression(max_iter=200,random_state=222)
model

model.fit(X_train,y_train)

y_predic=model.predict(X_test)
print(y_predic)
model.predict_proba(X_test)

print("Accuracy of the model is :  %3f " %
accuracy_score(y_test,y_predic))

print(confusion_matrix(y_test,y_predic))

print(classification_report(y_test, y_predic))
```

```
#applying decision tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
X_train, X_test, y_train, y_test = train_test_split(x,
y,test_size=0.2,random_state=2)

# Create object from the decision tree function
clf = DecisionTreeClassifier()

# make a pipeline for normalisation
pipeline_DT = Pipeline([('scaler', StandardScaler()), ('clf', clf)])

# Fit the data
fit = pipeline_DT.fit(X_train, y_train)

# Predict the data
y_pre = fit.predict(X_test)

print("Accuracy of the model is :  %3f " % accuracy_score(y_test,y_pre))

print(confusion_matrix(y_test,y_pre))

print(classification_report(y_test, y_pre))

#applying knn
# Import KNN model function
from sklearn.neighbors import KNeighborsClassifier

# Create list to contain accuracy values
acc_knn = []

#Create function that produces an accuracy score - to feed into for loop
below
def knn(k = 3):
    clf_knn = KNeighborsClassifier(k)
    fit_knn = clf_knn.fit(X_train, y_train)
    predicted_knn = fit_knn.predict(X_test)
    cm_knn = confusion_matrix(y_test, predicted_knn)
```

```
    acc_sc = accuracy_score(y_test, predicted_knn)
    return acc_sc

k_values = []
acc_values = []

# Loop through k values, stepping by 2
for k in range(1,100, 2):
    acc_sc = knn(k)
    k_values.append(k)
    acc_values.append(acc_sc)

# Create dataframe to with k values and the respective accuracy
acc_knn = pd.DataFrame({
    "k": k_values,
    "acc": acc_values
})

acc_knn.plot(x="k", y="acc")

# Find maximum accuracy value
acc_knn[acc_knn['acc'] == acc_knn['acc'].max()].k

# Display accuracy values for the different k values used
acc_knn.sort_values('acc',ascending=False)

# Build train and test model
X_train, X_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size = 0.2,
                                                    random_state=3)
# Set K value
clf_knn = KNeighborsClassifier(3)

# make a pipeline for normalisation
pipeline_knn = Pipeline([('scaler', StandardScaler()), ('knn', clf_knn)])

# Fit the data
fit_knn = pipeline_knn.fit(X_train, y_train)
```

```
# Predict on unseen data
y_predicted = fit_knn.predict(X_test)

# View Confusion matrix
cm_knn = confusion_matrix(y_test, y_predicted)
cm_knn

report_knn = classification_report(y_test, y_predicted)
print(report_knn)
```