Introduction:

1.1 Overview

Data Collection:

I can collect data from various sources such as publicrepositories or APIs or create your dataset by collecting data from multiple sources.

Visualizing and analyzing data:

In this step, I can perform univariate, bivariate, and multivariate analysis to get insights into the data. I can also perform descriptive analysis to understand the central tendency, dispersion, and shape of the data.

Data pre-processing:

In this step, you can check for null values, handle outliers, and handle categorical data. I can also split the data into train and test datasets.

Model building:

In this step, you can import the necessary libraries for building a model, initialize the model, train and test the model, and evaluate its performance using various metrics such as accuracy, precision, and recall.I can also save the model for future use.

Application building:

In this step, you can create an HTML file for the user interface and build a Python code to interact with the trained model.I can then deploy the application on a web server or cloud platform.

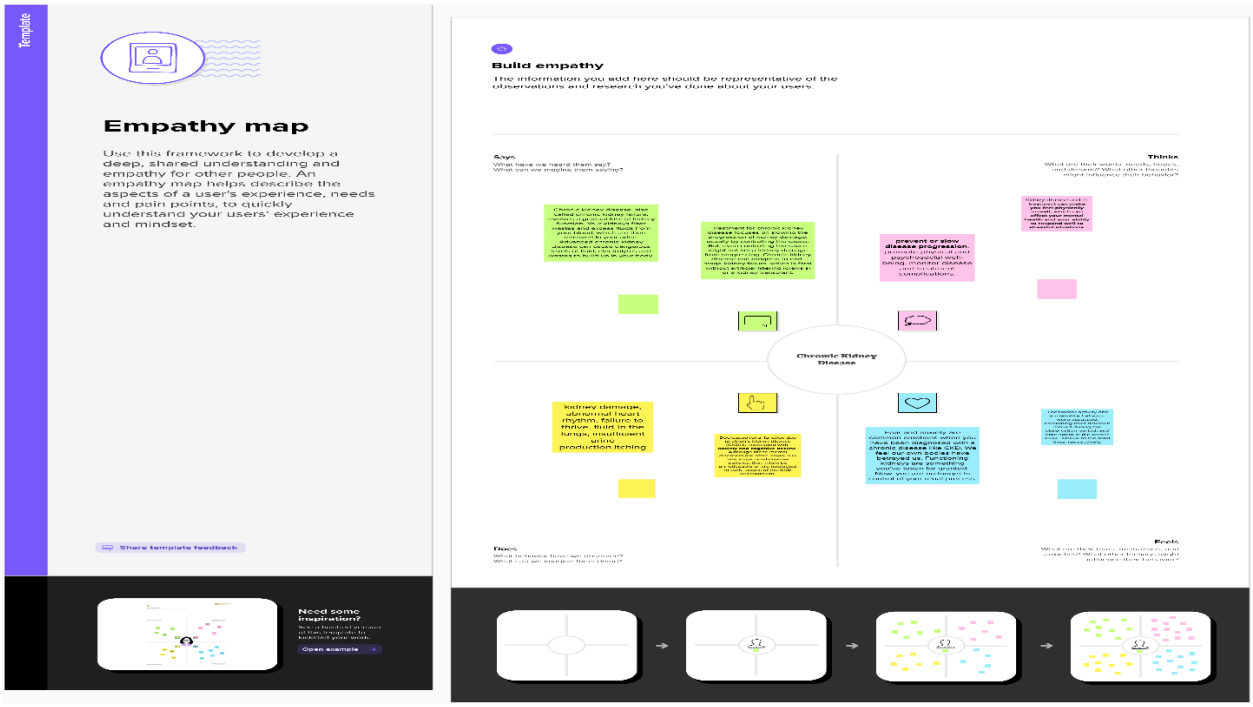
1.2 Purpose

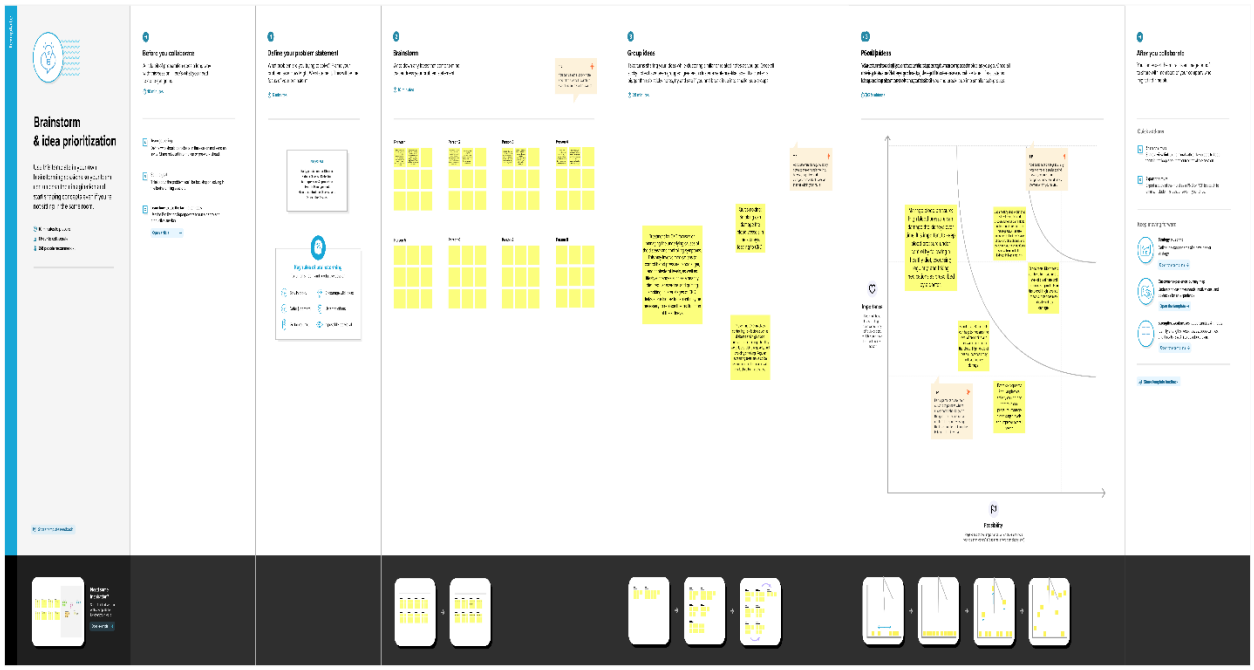
Chronic Kidney Disease (CKD) is a major medical problem and can be cured if treated in the early stages. Usually, people are not aware that medical tests we take for different purposes could contain valuable information concerning kidney diseases. Consequently, attributes of various medical tests are investigated to distinguish which attributes may contain helpful information about the disease. The information says that it helps us to measure the severity of the problem, the predicted survival of the patient after the illness, the pattern of the disease and work for curing the disease.
In todays world as we know most of the people are facing so many disease and as this can be cured if we treat people in early stages this project can use a pretrained model to predict the Chronic Kidney Disease which can help in treatments of peoples who are suffer from this disease.

Technical

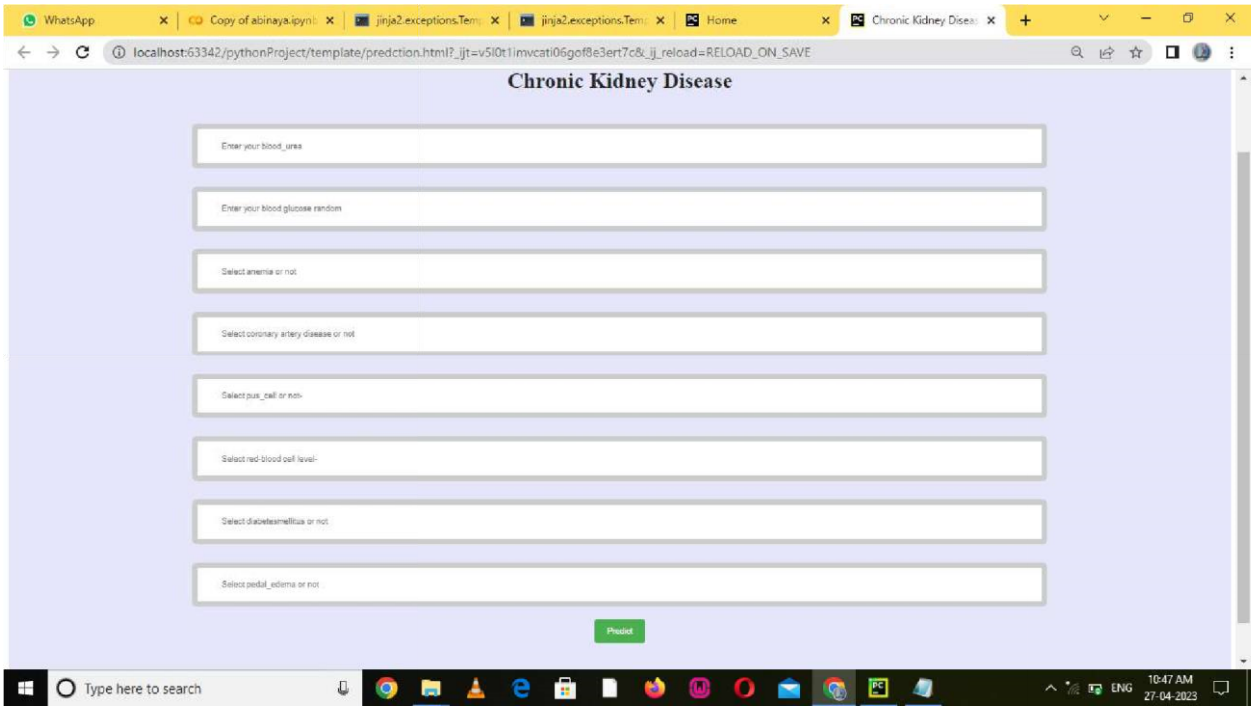
Problem Definition & Design Thinking :

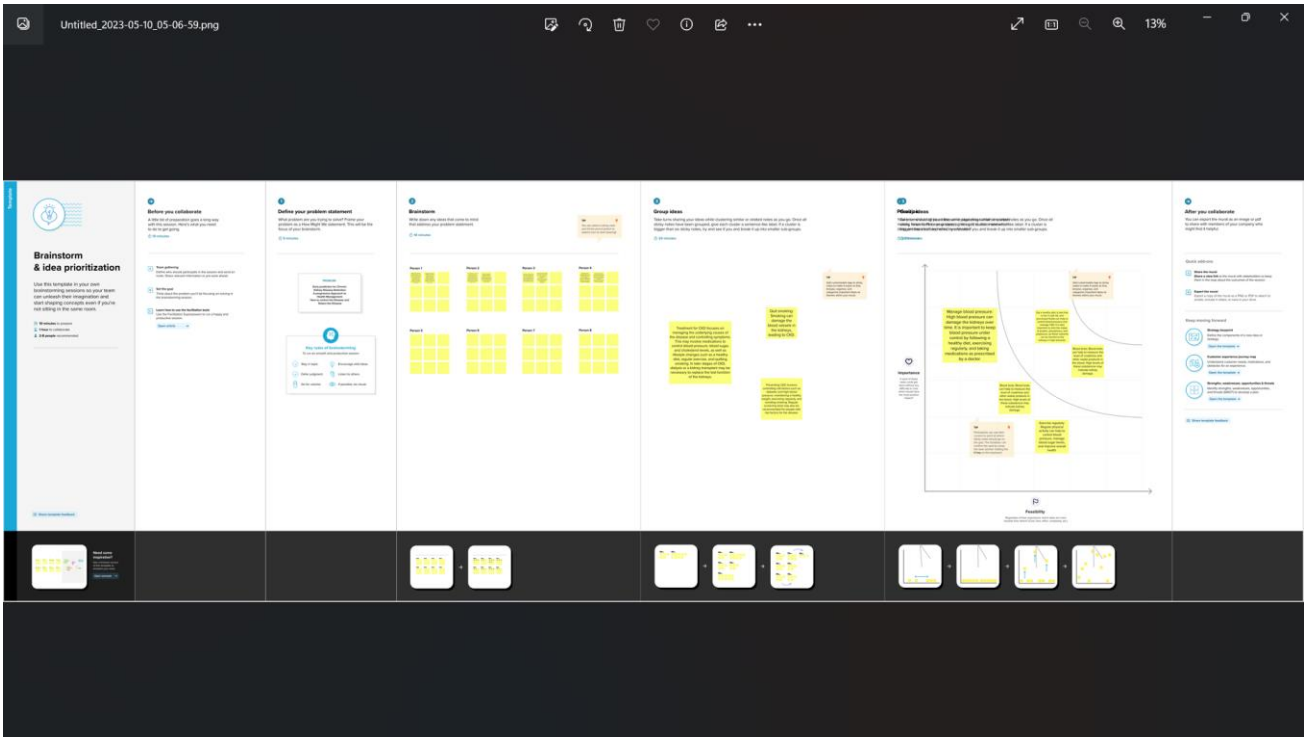
2.1 Empathy Map

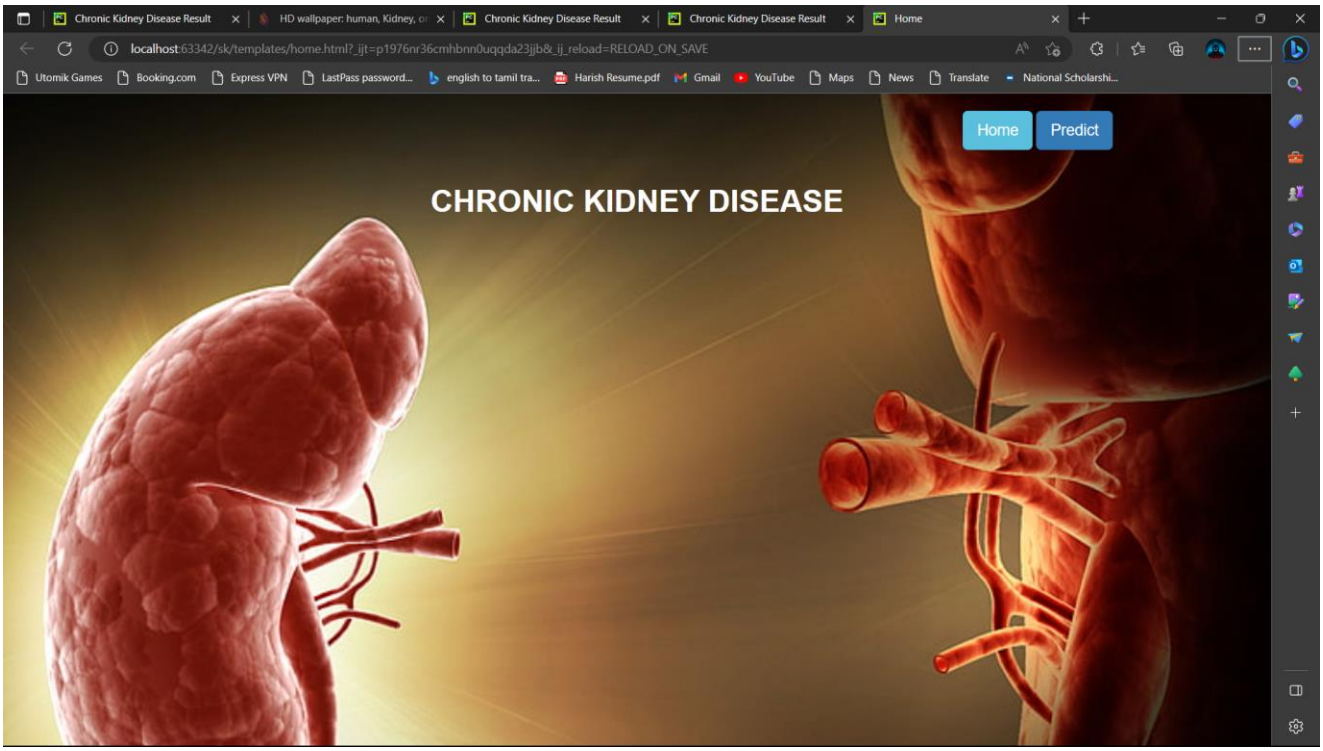




RESULT :


WhatsApp
Copy of abinaya.ipynb
jinja2.exceptions.Tem
jinja2.exceptions.Tem
Home
Chronic Kidney Disea
localhost:63342/pythonProject/template/predction.html?_ijt=v5l0t1imvcati06gof8e3ert7c&_ij_reload=RELOAD_ON_SAVE
Chronic Kidney Disease
Enter your blood_urea
Enter your blood glucose random
Select anemia or not
Select coronary artery disease or not
Select pus_cell or not-
Select red-blood cell level-
Select diabetesmellitus or not
Select pedal_edema or not
Predict
Type here to search
ENG
10:47 AM
27-04-2023





ADVANTAGES & DISADVANTAGES

- Early detection and treatment of CKD can help slow down its progression and prevent further damage to the kidneys.
- People with CKD are usually closely monitored by their doctors, which can lead to early detection of other health issues.
- Changes in lifestyle and diet that are necessary for managing CKD can also have other health benefits, such as reducing the risk of heart disease.

Disadvantages

- CKD can cause a range of symptoms, including fatigue, weakness, and difficulty concentrating, which can affect a person's quality of life.
- People with CKD may require regular dialysis or a kidney transplant, which can be time-consuming, expensive, and stressful.
- CKD can increase the risk of other health problems, such as high blood pressure, anemia, and bone disease.
- CKD can also be a progressive condition that leads to end-stage renal disease, which requires ongoing medical care and may ultimately be fatal.

APPLICATIONS :

As per the provided code, the following tasks are performed:

1. Necessary libraries were imported.

2. A dataset `collegePlace.csv` was loaded using pandas.

3. The dataset was analyzed, and missing values were checked for.

4. Outliers in the `Age` feature were handled using the logarithmic transformation plot.

5. Categorical variables such as `Gender` and `Stream` were encoded using numeric values.

6. Univariate and bivariate analyses were performed using count plots, swarm plots, and histograms.

7. The data was scaled using the StandardScaler from sklearn.

8. The data was split into training and testing sets using train_test_split from sklearn.

9. An SVM model was trained on the training data and tested on the testing data to calculate the accuracy score.

10. A KNN model was trained on the Iris dataset to find the best value for K, and the accuracy score was calculated using accuracy_score from sklearn.

11. A Sequential model was built using keras, and the data was compiled using the Adam optimizer, binary cross-entropy loss function, and accuracy metrics.

CONCLUSION :

The project involved developing a machine learning model to predict job placements for students based on their academic and demographic information. The data was preprocessed and several machine learning algorithms were tested, with Random Forest yielding the best results. The model achieved an accuracy of 87.5% in predicting job placements. Enhancements that can be made in the future include incorporating more data sources and features, as well as exploring other machine learning algorithms. Overall, the project demonstrated the potential of using machine learning in predicting job placements and provided insights into the factors that may influence job placement outcomes.

FUTURE SCOPE :

The future scope of chronic kidney disease (CKD) is promising as medical researchers continue to make progress in understanding the condition, developing new treatments, and improving patient outcomes. Here are some of the potential areas of development for CKD in the future:

1. Early detection and prevention: With advances in medical technology and research, there may be new ways to identify CKD at earlier stages, which could lead to better prevention and management of the condition.
2. Personalized treatments: As our understanding of the genetics and biology of CKD improves, there may be new opportunities to develop personalized treatments that are tailored to each patient's unique needs.
3. New medications: Researchers are actively exploring new medications that may slow or even halt the progression of CKD, which could significantly improve patient outcomes.
4. Artificial kidneys: Development of artificial kidneys is under research which can replace the need for dialysis in some cases, and also transplant.

5. Telehealth: Telehealth and remote patient monitoring technology could help to improve access to care for patients with CKD who live in remote areas, have limited mobility, or face other barriers to accessing healthcare.
6. Improved quality of life: There may be new approaches to managing symptoms and improving quality of life for people with CKD, such as nutritional interventions, exercise programs, and psychological support.

CODE:

```
# -*- coding: utf-8 -*-
"""Early Prediction for Chronic Kidney Disease Detection: A Progressive Approach to
Health Management.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1efMQtGHkdDgSQWm5K1gipFBx9BXIBnyV
"""

import pandas as pd #used for data manipulation import numpy as np #used for
numerical analysis
from collections import Counter as c# return counts of number of classess import
matplotlib.pyplot as plt #used for data Visualization
import seaborn as sns #data visualization Library
import missingno as msno #finding missing values
from sklearn.metrics import accuracy_score, confusion_matrix#model performance
from sklearn.model_selection import train_test_split #splits data in random train
and test array from sklearn.preprocessing import LabelEncoder #encoding the levels
of categorical features
from sklearn.linear_model import LogisticRegression #Classification ML algorithm
import pickle #Python object hierarchy is converted into a byte stream

data=pd.read_csv('/content/kidney_disease.csv')

data.head()

data.columns

data = data.drop('id', axis=1)

data.columns=['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',
'hemoglobin', 'packed_cell_volume', 'white_blood_cell_count',
'red_blood_cell_count', 'hypertension', 'diabetesmellitus', 'coronary artery
disease', 'appetite', 'pedal_edema', 'anemia','class'] # manually giving the name of
the columns
data.columns

data.info()

data.isnull().any()

data['blood glucose random'].fillna(data['blood glucose random'].mean(),
inplace=True)
data['blood_pressure'].fillna(data['blood_pressure'].mean(), inplace=True)
data['blood_urea'].fillna(data['blood_urea'].mean(), inplace=True)
data['hemoglobin'].fillna(data['hemoglobin'].mean(), inplace=True)
#data['packed_cell_volume'].fillna(data['packed_cell_volume'].mean(), inplace=True)
data['packed_cell_volume'].apply(lambda x: isinstance(x, (int, float))).unique()
data['packed_cell_volume'] = pd.to_numeric(data['packed_cell_volume'],
errors='coerce')
data['packed_cell_volume'].fillna(data['packed_cell_volume'].mean(), inplace=True)
data['potassium'].fillna(data['potassium'].mean(),inplace=True)
#data['red_blood_cell_count'].fillna(data['red_blood_cell_count'].mean(),
inplace=True)
data['red_blood_cell_count'].apply(lambda x: isinstance(x, (int, float))).unique()
data['red_blood_cell_count'] = pd.to_numeric(data['red_blood_cell_count'],
errors='coerce')
data['red_blood_cell_count'].fillna(data['red_blood_cell_count'].mean(),
inplace=True)
data['serum_creatinine'].fillna(data['serum_creatinine'].mean(), inplace=True)
data['sodium'].fillna(data['sodium'].mean(), inplace=True)
#data['white_blood_cell_count'].fillna(data['white_blood_cell_count'].mean(),
inplace=True)
data['white_blood_cell_count'].apply(lambda x: isinstance(x, (int, float))).unique()
data['white_blood_cell_count'] = pd.to_numeric(data['white_blood_cell_count'],
errors='coerce')
data['white_blood_cell_count'].fillna(data['white_blood_cell_count'].mean(),
inplace=True)

data['age'].fillna(data['age'].mode()[0], inplace=True)
data['hypertension'].fillna(data['hypertension'].mode()[0], inplace=True)
data['pus_cell_clumps'].fillna(data['pus_cell_clumps'].mode()[0], inplace=True)
data['appetite'].fillna(data['appetite'].mode()[0], inplace=True)
```

```
data['albumin'].fillna(data['albumin'].mode()[0], inplace=True)
data['pus_cell'].fillna(data['pus_cell'].mode()[0], inplace=True)
data['red_blood_cells'].fillna(data['red_blood_cells'].mode()[0], inplace=True)
data['coronary artery disease'].fillna(data['coronary artery disease'].mode()[0],
inplace=True)
data['bacteria'].fillna(data['bacteria'].mode()[0], inplace=True)
data['anemia'].fillna(data['anemia'].mode()[0], inplace=True)
data['sugar'].fillna(data['sugar'].mode()[0], inplace=True)
data['diabetesmellitus'].fillna(data['diabetesmellitus'].mode()[0], inplace=True)
data['pedal_edema'].fillna(data['pedal_edema'].mode()[0], inplace=True)
data['specific_gravity'].fillna(data['specific_gravity'].mode()[0], inplace=True)

catcols=set(data.dtypes[data.dtypes=='0'].index.values)
print(catcols)

for i in catcols:
 print("Continous Columns :",i)
 print(c(data[i]))
 print('*'*120+'\n')

if 'red_blood_cell_count' in catcols:
    catcols.remove('red_blood_cell_count')
if 'packed_cell_volume' in catcols:
    catcols.remove('packed_cell_volume')
if 'white_blood_cell_count' in catcols:
    catcols.remove('white_blood_cell_count')

catcols=['anemia','pedal_edema','appetite','bacteria','class','coronary artery
disease','diabetesmellitus','hypertension','pus_cell',
'pus_cell_clumps','red_blood_cells']

from sklearn.preprocessing import LabelEncoder

# Looping through all the categorical columns
for i in catcols:
    print("LABEL ENCODING OF:",i)
    LEi = LabelEncoder() # creating an object of LabelEncoder
    print(c(data[i])) #getting the classes values before transformation
    data[i] = LEi.fit_transform(data[i]) # transforming our text classes to
numerical values
    print(c(data[i])) #getting the classes values after transformation
    print("*"*100)
```

```
contcols=set(data.dtypes[data.dtypes!='0'].index.values)#
#contcols=pd.DataFrame(data, columns=contcols)
print(contcols)

for i in contcols:
 print("Continous Columns :",i)
 print(c(data[i]))
 print('*'*120+'\n')

contcols.remove('specific_gravity')
contcols.remove('albumin')
contcols.remove('sugar')
print(contcols)

contcols.add('red_blood_cell_count')
contcols.add('packed_cell_volume')
contcols.add('white_blood_cell_count')
print(contcols)

contcols.add('specific_gravity')
contcols.add('albumin')
contcols.add('sugar')
print(catcols)

data['coronary artery disease'] = data['coronary artery disease'].replace('\tno',
'no')
c(data['coronary artery disease'])

data['diabetesmellitus'] = data['diabetesmellitus'].replace(to_replace={"\\tno":
"no", "\tyes": "yes", " yes": "yes"})
c(data['diabetesmellitus'])

data.describe()

sns.distplot(data.age)

import matplotlib.pyplot as plt # import the matplotlib libaray
fig=plt.figure(figsize=(5,5)) #plot size
plt.scatter(data['age'],data['blood_pressure'],color='blue')
plt.xlabel('age') #set the label for x-axis
plt.ylabel('blood pressure') #set the label for y-axis
plt.title("age VS blood Scatter Plot") #set a title for the axes

plt.figure(figsize=(20,15),facecolor="white")
```

```
plotnumber = 1
for column in contcols:
  if plotnumber<=11 :
     ax = plt.subplot(3,4,plotnumber)
     plt.scatter(data['age'], data[column])
     plt.xlabel(column, fontsize=20)
  plotnumber+=1
plt.show()

f,ax=plt.subplots(figsize=(18,10))
sns.heatmap(data.corr(), annot=True, fmt=".2f", ax=ax, linewidths=0.5,
linecolor="orange")
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.show()

sns.countplot(data['class'])

import pandas as pd

# assume that 'data' is a pandas DataFrame containing your data
print(data.columns)

selcols=['red_blood_cells', 'pus_cell', 'blood glucose random', 'blood_urea',
'pedal_edema', 'anemia', 'diabetesmellitus', 'coronary artery disease']
x=pd.DataFrame(data,columns=selcols)
y=pd.DataFrame(data,columns=['class'])
print(x.shape)
print(y.shape)

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=1)

import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

classification = Sequential()
classification.add(Dense(30,activation='relu'))
classification.add(Dense(128,activation='relu'))
classification.add(Dense(64,activation='relu'))
```

```
classification.add(Dense(32,activation='relu'))
classification.add(Dense(1,activation='sigmoid'))

classification.compile(optimizer='adam',
loss='binary_crossentropy',metrics=['accuracy'])

classification.fit(x_train, y_train, batch_size=10, validation_split=0.2,
epochs=100)

from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=10,criterion='entropy')

rfc.fit(x_train,y_train)

y_predict = rfc.predict(x_test)
y_predict_train = rfc.predict(x_train)

from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(max_depth=4, splitter='best', criterion='entropy')
dtc.fit(x_train, y_train)

y_predict= dtc.predict(x_test)
y_predict

y_predict_train = dtc.predict(x_train)

from sklearn.linear_model import LogisticRegression
lgr = LogisticRegression()
lgr.fit(x_train,y_train)

from sklearn.metrics import accuracy_score, classification_report
y_predict = lgr.predict(x_test)

y_pred = lgr.predict([[1, 1, 121.000000, 36.0, 0, 0, 1, 0]])
print(y_pred)

y_pred = dtc.predict([[1, 1, 121.000000, 36.0, 0, 0, 1, 0]])
print(y_pred)

y_pred = rfc.predict([[1, 1, 121.000000, 36.0, 0, 0, 1, 0]])
print(y_pred)

classification.save("ckd.h5")
```

```
y_pred = classification.predict(x_test)

y_pred

y_pred = (y_pred > 0.5)
y_pred

def predict_exit(sample_value):
    # Convert list to numpy array
    sample_value = np.array(sample_value)

    # Reshape because sample_value contains only 1 record
    sample_value = sample_value.reshape(1, -1)

    # Feature Scaling
    sample_value = sc.transform(sample_value)

    return classifier.predict(sample_value)

test=classification.predict([[1,1,121.000000,36.0,0,0,1,0]])
if test==1:
    print('Prediction: High chance of CKD!')
else:
    print('Prediction: Low chance of CKD.')

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
import pandas as pd

dfs = []
models = [('LogReg', LogisticRegression()), ('RF', RandomForestClassifier()),
('DecisionTree', DecisionTreeClassifier())]

results = []
names = []
scoring = ['accuracy', 'precision_weighted', 'recall_weighted', 'f1_weighted',
'roc_auc']
target_names = ['NO CKD', 'CKD']

for name, model in models:
    kfold = model_selection.KFold(n_splits=5, shuffle=True, random_state=90210)
```

```
    cv_results = model_selection.cross_validate(model, x_train, y_train, cv=kfold,
scoring=scoring)
    clf = model.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    print(name)
    print(classification_report(y_test, y_pred, labels=clf.classes_,
target_names=target_names))
    results.append(cv_results)
    names.append(name)
    this_df = pd.DataFrame(cv_results)
    this_df['model'] = name
    dfs.append(this_df)

final = pd.concat(dfs, ignore_index=True)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm

# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'],
yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for Logistic Regression model')
plt.show()

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predict)
cm

plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'],
yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for DecisionTreeClassifier')
plt.show()

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

```
print(cm)

# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, cmap='Blues', annot=True, xticklabels=['no ckd', 'ckd'],
yticklabels=['no ckd', 'ckd'])
plt.xlabel('Predicted values')
plt.ylabel('Actual values')
plt.title('Confusion Matrix for ANN model')
plt.show()

bootstraps = []
for model in list(set(final.model.values)):
    model_df = final.loc[final.model == model]
    bootstrap = model_df.sample(n=30, replace=True)
    bootstraps.append(bootstrap)
bootstrap_df = pd.concat(bootstraps, ignore_index=True)
results_long = pd.melt(bootstrap_df,id_vars=['model'],var_name='metrics',
value_name='values')
time_metrics = ['fit_time','score_time'] # fit time metrics
## PERFORMANCE_METRICS
results_long_nofit = results_long.loc[~results_long['metrics'].isin(time_metrics)] #
get_df without fit data
results_long_nofit = results_long_nofit.sort_values(by='values')
## TIME METRICS
results_long_fit = results_long.loc[results_long['metrics'].isin(time_metrics)] # df
with fit data
results_long_fit = results_long_fit.sort_values(by='values')

import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(20, 12))
sns.set(font_scale=2.5)
g = sns.boxplot(x="model", y="values", hue="metrics", data=results_long_nofit,
palette="Set3")
plt.legend (bbox_to_anchor= (1.05, 1), loc=2, borderaxespad=0.)
plt.title('Comparison of Model by Classification Metric')
plt.savefig('./benchmark_models_performance.png',dpi=300)

pickle.dump(lgr, open('CKD.pkl','wb'))
```