```
!pip install -q keras
import keras
```

```
import pandas as pd;
import numpy as np;
import io
```

```
from google.colab import files
```

```
uploaded = files.upload()
```

Choose Files chronic_kid…disease.csv

- **chronic_kidney_disease.csv**(text/csv) - 29325 bytes, last modified: 10/24/2020 - 100% done

```
Saving chronic_kidney_disease.csv to chronic_kidney_disease (5).csv
```

```
from sklearn.model_selection import train_test_split
from sklearn.impute import KNNImputer
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

#Create function for checking missing values which accepts a dataframe as its parameter
def null_values_check(df):
    #Error handling to prevent abnormal termination of operation
    try:
        #if-else statement for null value check
        if(df.isnull().values.any() == True):
            #if there are null values present, print a column-wise summary of records with null values
            print('Number of null records within each column:\n' + str(df.isnull().sum()))
        else:
            print('There is no missing values in the dataset.')

    except Exception as e:
```

```
        logging.error(e)
dataset_name = 'kidney_disease.csv'

#error-handling to prevent abnormal termination of code
try:
    #import and load weather dataset into pandas dataframe
    chronic_kidney_disease_dataframe = pd.read_csv(io.BytesIO(uploaded['chronic_kidney_disease.csv']))

    #Description of Datasets
    #Print number of records and attributes of whole kidney dataset
    print('Shape of dataset: ' + str(chronic_kidney_disease_dataframe.shape))
    print('Total number of records in dataset = ' + str(chronic_kidney_disease_dataframe.shape[0]))
    print('Total number of attributes in dataset = ' + str(chronic_kidney_disease_dataframe.shape[1]))
    print('')
    #call function created to check for null values
    null_values_check(chronic_kidney_disease_dataframe)
    #Missing value imputation
    #replace ? to nan values
    chronic_kidney_disease_dataframe = chronic_kidney_disease_dataframe.replace('?', np.nan)

    #set the features and the target variables
    target_class = chronic_kidney_disease_dataframe['class']
    print('\nAre there missing values in Target Class? ' + str(target_class.isna().any()))
    feature_classes = chronic_kidney_disease_dataframe.iloc[:, 0:24]
    print('\nAre there missing values in the Features? \n' + str(feature_classes.isna().any()))

    #KNN imputation (n_neighbour = 5 means that the missing values will be replaced by the mean value of 5 nearest neighbors
    knn_missing_values_imputer = KNNImputer(n_neighbors=5)
    feature_classes = pd.DataFrame(knn_missing_values_imputer.fit_transform(feature_classes),
                                   columns = feature_classes.columns)
    print('\nNow, Are there any missing values in Features? ' + str(feature_classes.isna().any()))


    #Scaling and normalization of features
    standard_feature_scaler = StandardScaler()
    feature_classes = standard_feature_scaler.fit_transform(feature_classes)
    feature_classes = pd.DataFrame(feature_classes, columns=['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc',
                                                             'pcc', 'ba', 'bgr', 'bu', 'sc', 'sod', 'pot',
```

```
                                                                    'hemo', 'pcv', 'wbcc', 'rbcc', 'htn', 'dm',
                                                                    'cad', 'appet', 'pe', 'ane'])

    #Encoding target class using label encoding
    target_label_encoder = preprocessing.LabelEncoder()
    target_class = target_label_encoder.fit_transform(target_class)
    target_class1 = pd.DataFrame(target_class, columns=['class'])

    #split the dataset into training and testing data
    train_features, test_features, train_target, test_target = train_test_split(feature_classes, target_class,
                                                                                 train_size = 0.7, test_size = 0.3)
    print('\nAfter Pre-processing:')
    print('Size of train dataset: ' + str(train_target.shape[0]))
    print('Size of test dataset: ' + str(test_target.shape[0]))

except FileNotFoundError as e:
    logging.error(e)
```

```
Are there missing values in the Features?
age       True
bp        True
sg        True
al        True
su        True
rbc      False
pc       False
pcc      False
ba       False
bgr       True
bu        True
sc        True
sod       True

pot       True
hemo      True
pcv      False
wbcc     False
rbcc     False
htn      False
dm       False
```

```
cad      False
appet    False
pe       False
ane      False
dtype: bool

Now, Are there any missing values in Features? age      False
bp       False
sg       False
al       False
su       False
rbc      False
pc       False
pcc      False
ba       False
bgr      False
bu       False
sc       False
sod      False
pot      False
hemo     False
pcv      False
wbcc     False
rbcc     False
htn      False
dm       False
cad      False
appet    False
pe       False
ane      False
dtype: bool

After Pre-processing:
Size of train dataset: 280
Size of test dataset: 120
```

```
from collections import Counter
from imblearn.over_sampling import SMOTE
counter=Counter(train_target)
print('before',counter)
smt=SMOTE()
```

```
x_train_sm,y_train_sm=smt.fit_resample(train_features,train_target)
counter=Counter(y_train_sm)
print('after',counter)
```

```
before Counter({0: 182, 1: 98})
after Counter({0: 182, 1: 182})
```

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report
```

```
linear_svm=SVC(C=1.0,kernel='linear',random_state=0)
linear_svm=linear_svm.fit(x_train_sm,y_train_sm)
```

```
print(linear_svm.score(x_train_sm,y_train_sm))
print(linear_svm.score(test_features,test_target))
```

```
0.9972527472527473
0.9666666666666667
```

```
predicted =linear_svm.predict(test_features)
report = classification_report(test_target, predicted)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        68
           1       0.96      0.96      0.96        52

    accuracy                           0.97       120
   macro avg       0.97      0.97      0.97       120
weighted avg       0.97      0.97      0.97       120
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
clf = RandomForestClassifier(n_estimators = 100)

# Training the model on the training dataset
# fit function is used to train the model using the training sets as parameters
clf.fit(x_train_sm, y_train_sm)

# performing predictions on the test dataset
y_pred = clf.predict(test_features)

# metrics are used to find accuracy or error
from sklearn import metrics
print()

# using metrics module for accuracy calculation
print("ACCURACY OF THE MODEL: ", clf.score(test_features,test_target))
```

```
ACCURACY OF THE MODEL:  0.9666666666666667
```

✓ 0s completed at 10:00