Name: Gullas Rainer L.

Section: BSCPE32S3

Date Perfomred: 04/07/2024

Date Submitted: 04/11/2024

Instructor: Engr. Roman Richard

In this activity I used the datasets wine for classification [Wine - UCI] the problem that is/are being addressed is that people want to know here the origin of their wine or in short where it came from. the second one is the communities and crimes [Communities and Crime - UCI] the problem that is/are being addressed here is the data combines socio-economic data from the 1990 US Census, law enforcement data from the 1990 US LEMAS survey, and crime data from the 1995 FBI UCR.

Classification

```
!pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.10/dist-packages (0.0.6)
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_auc_score, roc_curve, accuracy_score
from sklearn.ensemble import RandomForestClassifier

import seaborn as sns
%matplotlib inline

from keras.models  import Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.optimizers import Adam, SGD, RMSprop
```

```
from ucimlrepo import fetch_ucirepo

wine = fetch_ucirepo(id=109)

X = wine.data.features
y = wine.data.targets

print(wine.metadata)
print(wine.variables)
```

```
{'uci_id': 109, 'name': 'Wine', 'repository_url': 'https://archive.ics.uci.edu/dataset/109/wine', 'data_url': 'https://archive.ics.uci.e
                            name     role         type demographic  \
0                          class   Target  Categorical        None
1                        Alcohol  Feature   Continuous        None
2                      Malicacid  Feature   Continuous        None
3                            Ash  Feature   Continuous        None
4              Alcalinity_of_ash  Feature   Continuous        None
5                      Magnesium  Feature      Integer        None
6                  Total_phenols  Feature   Continuous        None
7                     Flavanoids  Feature   Continuous        None
8           Nonflavanoid_phenols  Feature   Continuous        None
9                Proanthocyanins  Feature   Continuous        None
10               Color_intensity  Feature   Continuous        None
11                           Hue  Feature   Continuous        None
12  0D280_0D315_of_diluted_wines  Feature   Continuous        None
13                       Proline  Feature      Integer        None
```

```
   description units missing_values
0         None  None             no
1         None  None             no
2         None  None             no
3         None  None             no
4         None  None             no
5         None  None             no
6         None  None             no
7         None  None             no
8         None  None             no
9         None  None             no
10        None  None             no
11        None  None             no
12        None  None             no
13        None  None             no
```

```
1 X
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins	Color_inte
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	
...	...	...	...	...	...	...	...	...	...	
173	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	
174	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	
175	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	
176	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	
177	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	

178 rows × 13 columns

```
1 y
```

	class
0	1
1	1
2	1
3	1
4	1
...	...
173	3
174	3
175	3
176	3
177	3

178 rows × 1 columns

```
1 missing_values = X.isnull().sum()
2
3 print(missing_values)
```

```
Alcohol                         0
Malicacid                       0
Ash                             0
Alcalinity_of_ash               0
Magnesium                       0
Total_phenols                   0
Flavanoids                      0
```

```
Nonflavanoid_phenols            0
Proanthocyanins                 0
Color_intensity                 0
Hue                             0
0D280_0D315_of_diluted_wines    0
Proline                         0
dtype: int64
```

```
X.sample(5)
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins	Color_inten
57	13.29	1.97	2.68	16.8	102	3.00	3.23	0.31	1.66	
60	12.33	1.10	2.28	16.0	101	2.05	1.09	0.63	0.41	
92	12.69	1.53	2.26	20.7	80	1.38	1.46	0.58	1.62	
65	12.37	1.21	2.56	18.1	98	2.42	2.65	0.37	2.08	
88	11.64	2.06	2.46	21.6	84	1.95	1.69	0.48	1.35	

```
X_1 = X.iloc[:, 0].values
y_1 = y["class"].values
```

```
import pandas as pd
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

model = RandomForestClassifier(random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)

kfold = KFold(n_splits=10, shuffle=True, random_state=42)
scores = cross_val_score(model, X, y, cv=kfold)
print("Cross-validated Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

```

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed
  estimator.fit(X_train, y_train, **fit_params)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:686: DataConversionWarning: A column-vector y was passed
  estimator.fit(X_train, y_train, **fit_params)
<ipython-input-393-a65b9b25cef4>:14: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  model.fit(X_train, y_train)
Cross-validated Accuracy: 0.99 (+/- 0.04)
Accuracy: 0.97

Classification Report:
              precision    recall  f1-score   support

           1       0.93      1.00      0.97        14
           2       1.00      0.92      0.96        13
```

```
           3       1.00      1.00      1.00         9

    accuracy                           0.97        36
   macro avg       0.98      0.97      0.98        36
weighted avg       0.97      0.97      0.97        36
```

```
normalizer = StandardScaler()
X_train_norm = normalizer.fit_transform(X_train)
X_test_norm = normalizer.transform(X_test)
```

```
print("Shape of X_train_norm:", X_train_norm.shape)
print("Shape of X_test_norm:", X_test_norm.shape)
```

```
Shape of X_train_norm: (142, 13)
Shape of X_test_norm: (36, 13)
```

```
from keras.layers import GaussianNoise
input_shape = (X_train_norm.shape[1],)
model = Sequential([
    Dense(10, input_shape=input_shape, activation='relu'),
    GaussianNoise(0.5),
    Dense(3, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

```
model.summary()
```

```
Model: "sequential_65"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_207 (Dense)           (None, 10)                140

 gaussian_noise_3 (Gaussian  (None, 10)                0
 Noise)

 dense_208 (Dense)           (None, 3)                 33

 dense_209 (Dense)           (None, 1)                 4

=================================================================
Total params: 177 (708.00 Byte)
Trainable params: 177 (708.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

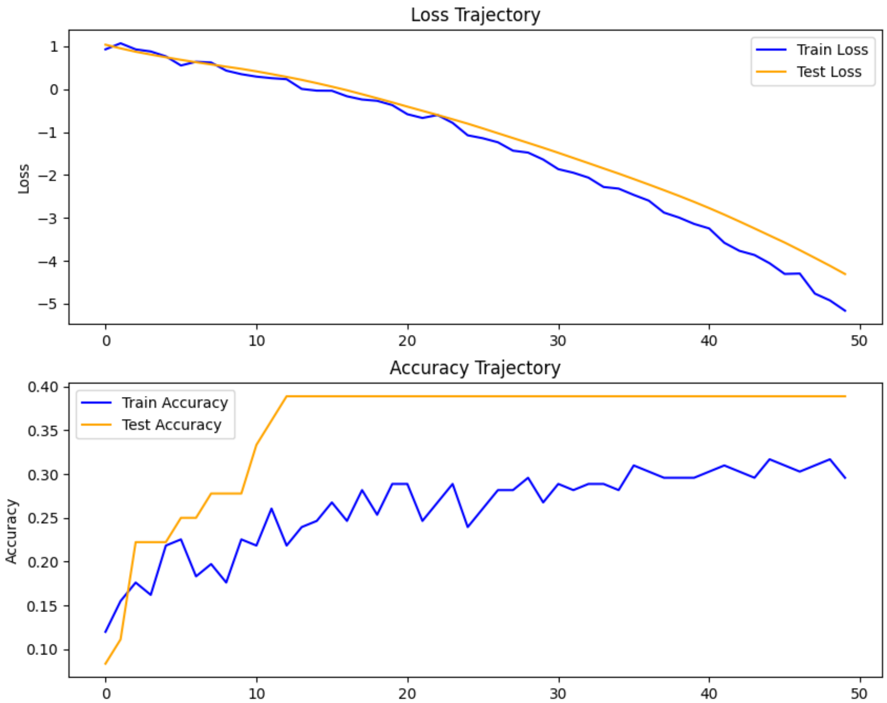
```
from keras.optimizers import Adam

optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

run_hist_1 = model.fit(X_train_norm, y_train, validation_data=(X_test_norm, y_test), epochs=50)

```

```
Epoch 1/50
5/5 [==============================] - 1s 49ms/step - loss: 0.9285 - accuracy: 0.1197 - val_loss: 1.0369 - val_accuracy: 0.0833
Epoch 2/50
5/5 [==============================] - 0s 13ms/step - loss: 1.0702 - accuracy: 0.1549 - val_loss: 0.9520 - val_accuracy: 0.1111
Epoch 3/50
5/5 [==============================] - 0s 14ms/step - loss: 0.9279 - accuracy: 0.1761 - val_loss: 0.8728 - val_accuracy: 0.2222
Epoch 4/50
5/5 [==============================] - 0s 11ms/step - loss: 0.8793 - accuracy: 0.1620 - val_loss: 0.8075 - val_accuracy: 0.2222
Epoch 5/50
5/5 [==============================] - 0s 10ms/step - loss: 0.7653 - accuracy: 0.2183 - val_loss: 0.7440 - val_accuracy: 0.2222
Epoch 6/50
5/5 [==============================] - 0s 9ms/step - loss: 0.5526 - accuracy: 0.2254 - val_loss: 0.6844 - val_accuracy: 0.2500
Epoch 7/50
5/5 [==============================] - 0s 14ms/step - loss: 0.6397 - accuracy: 0.1831 - val_loss: 0.6285 - val_accuracy: 0.2500
Epoch 8/50
5/5 [==============================] - 0s 14ms/step - loss: 0.6219 - accuracy: 0.1972 - val_loss: 0.5775 - val_accuracy: 0.2778
Epoch 9/50
5/5 [==============================] - 0s 14ms/step - loss: 0.4329 - accuracy: 0.1761 - val_loss: 0.5251 - val_accuracy: 0.2778
Epoch 10/50
5/5 [==============================] - 0s 15ms/step - loss: 0.3497 - accuracy: 0.2254 - val_loss: 0.4739 - val_accuracy: 0.2778
Epoch 11/50
5/5 [==============================] - 0s 10ms/step - loss: 0.2928 - accuracy: 0.2183 - val_loss: 0.4159 - val_accuracy: 0.3333
```

```
Epoch 12/50
5/5 [==============================] - 0s 10ms/step - loss: 0.2568 - accuracy: 0.2606 - val_loss: 0.3521 - val_accuracy: 0.3611
Epoch 13/50
5/5 [==============================] - 0s 14ms/step - loss: 0.2326 - accuracy: 0.2183 - val_loss: 0.2867 - val_accuracy: 0.3889
Epoch 14/50
5/5 [==============================] - 0s 9ms/step - loss: 0.0065 - accuracy: 0.2394 - val_loss: 0.2170 - val_accuracy: 0.3889
Epoch 15/50
5/5 [==============================] - 0s 9ms/step - loss: -0.0341 - accuracy: 0.2465 - val_loss: 0.1407 - val_accuracy: 0.3889
Epoch 16/50
5/5 [==============================] - 0s 13ms/step - loss: -0.0365 - accuracy: 0.2676 - val_loss: 0.0595 - val_accuracy: 0.3889
Epoch 17/50
5/5 [==============================] - 0s 11ms/step - loss: -0.1653 - accuracy: 0.2465 - val_loss: -0.0251 - val_accuracy: 0.3889
Epoch 18/50
5/5 [==============================] - 0s 10ms/step - loss: -0.2411 - accuracy: 0.2817 - val_loss: -0.1175 - val_accuracy: 0.3889
Epoch 19/50
5/5 [==============================] - 0s 13ms/step - loss: -0.2713 - accuracy: 0.2535 - val_loss: -0.2113 - val_accuracy: 0.3889
Epoch 20/50
5/5 [==============================] - 0s 13ms/step - loss: -0.3709 - accuracy: 0.2887 - val_loss: -0.3074 - val_accuracy: 0.3889
Epoch 21/50
5/5 [==============================] - 0s 10ms/step - loss: -0.5833 - accuracy: 0.2887 - val_loss: -0.4066 - val_accuracy: 0.3889
Epoch 22/50
5/5 [==============================] - 0s 13ms/step - loss: -0.6712 - accuracy: 0.2465 - val_loss: -0.5041 - val_accuracy: 0.3889
Epoch 23/50
5/5 [==============================] - 0s 11ms/step - loss: -0.6029 - accuracy: 0.2676 - val_loss: -0.6007 - val_accuracy: 0.3889
Epoch 24/50
5/5 [==============================] - 0s 11ms/step - loss: -0.7858 - accuracy: 0.2887 - val_loss: -0.7019 - val_accuracy: 0.3889
Epoch 25/50
5/5 [==============================] - 0s 9ms/step - loss: -1.0754 - accuracy: 0.2394 - val_loss: -0.8039 - val_accuracy: 0.3889
Epoch 26/50
5/5 [==============================] - 0s 9ms/step - loss: -1.1434 - accuracy: 0.2606 - val_loss: -0.9142 - val_accuracy: 0.3889
Epoch 27/50
5/5 [==============================] - 0s 9ms/step - loss: -1.2378 - accuracy: 0.2817 - val_loss: -1.0272 - val_accuracy: 0.3889
Epoch 28/50
5/5 [==============================] - 0s 13ms/step - loss: -1.4333 - accuracy: 0.2817 - val_loss: -1.1393 - val_accuracy: 0.3889
Epoch 29/50
5/5 [==============================] - 0s 9ms/step - loss: -1.4784 - accuracy: 0.2958 - val_loss: -1.2519 - val_accuracy: 0.3889
```

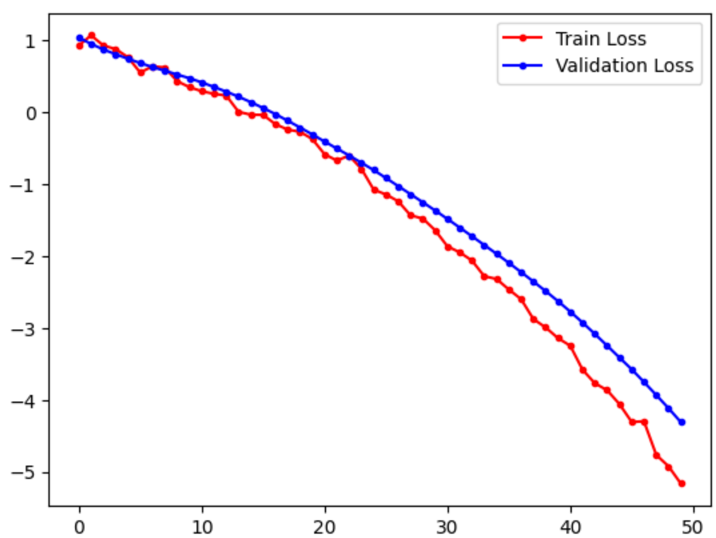
```python
train_loss = run_hist_1.history['loss']
test_loss = run_hist_1.history['val_loss']
train_accuracy = run_hist_1.history['accuracy']
test_accuracy = run_hist_1.history['val_accuracy']

fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))

axes[1].plot(train_accuracy, label='Train Accuracy', color='blue')
axes[1].plot(test_accuracy, label='Test Accuracy', color='orange')
axes[1].set_title('Accuracy Trajectory')
axes[1].set_ylabel('Accuracy')
axes[1].legend()

axes[0].plot(train_loss, label='Train Loss', color='blue')
axes[0].plot(test_loss, label='Test Loss', color='orange')
axes[0].set_title('Loss Trajectory')
axes[0].set_ylabel('Loss')
axes[0].legend()
```

```
<matplotlib.legend.Legend at 0x7832f2c45750>
```




```
1 fig, ax = plt.subplots()
2 ax.plot(run_hist_1.history["loss"],'r', marker='.', label="Train Loss")
3 ax.plot(run_hist_1.history["val_loss"],'b', marker='.', label="Validation Loss")
4 ax.legend()
```

```
<matplotlib.legend.Legend at 0x7832f2b4ff10>
```



Regression

```python
from ucimlrepo import fetch_ucirepo
import pandas as pd
from sklearn.model_selection import train_test_split

def get_data():
    communities_and_crime = fetch_ucirepo(id=183)

    X = communities_and_crime.data.features
    y = communities_and_crime.data.targets

    return X, y

def get_combined_data():
    X, y = get_data()
    X.drop(['population'], axis=1, inplace=True)
    combined = pd.concat([X, y], axis=1)

    return combined

combined_data = get_combined_data()
print(combined_data.head())

```

```
   state county community        communityname  fold  householdsize  \
0      8      ?         ?         Lakewoodcity     1           0.33
1     53      ?         ?          Tukwilacity     1           0.16
2     24      ?         ?         Aberdeentown     1           0.42
3     34      5     81440  Willingborotownship     1           0.77
4     42     95      6096    Bethlehemtownship     1           0.55

   racepctblack  racePctWhite  racePctAsian  racePctHisp  ...  LandArea  \
0          0.02          0.90          0.12         0.17  ...      0.12
1          0.12          0.74          0.45         0.07  ...      0.02
2          0.49          0.56          0.17         0.04  ...      0.01
3          1.00          0.08          0.12         0.10  ...      0.02
4          0.02          0.95          0.09         0.05  ...      0.04

   PopDens  PctUsePubTrans  PolicCars  PolicOperBudg  LemasPctPolicOnPatr  \
0     0.26            0.20       0.06           0.04                  0.9
1     0.12            0.45          ?              ?                    ?
2     0.21            0.02          ?              ?                    ?
3     0.39            0.28          ?              ?                    ?
4     0.09            0.02          ?              ?                    ?

   LemasGangUnitDeploy  LemasPctOfficDrugUn  PolicBudgPerPop  \
0                  0.5                 0.32             0.14
1                    ?                 0.00                ?
2                    ?                 0.00                ?
3                    ?                 0.00                ?
4                    ?                 0.00                ?

   ViolentCrimesPerPop
0                 0.20
1                 0.67
2                 0.43
3                 0.12
4                 0.03

[5 rows x 127 columns]
```

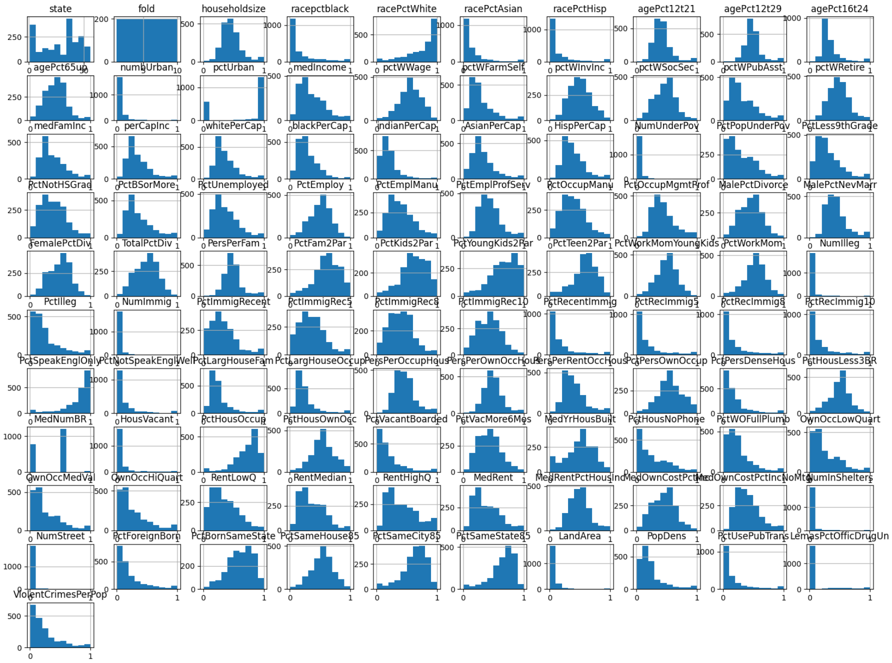
```
def get_cols_with_no_nans(df,col_type):
    '''
    Arguments :
    df : The dataframe to process
    col_type :
          num : to only get numerical columns with no nans
          no_num : to only get nun-numerical columns with no nans
          all : to get any columns with no nans
    '''
    if (col_type == 'num'):
        predictors = df.select_dtypes(exclude=['object'])
    elif (col_type == 'no_num'):
        predictors = df.select_dtypes(include=['object'])
    elif (col_type == 'all'):
        predictors = df
    else :
        print('Error : choose a type (num, no_num, all)')
        return 0
    cols_with_no_nans = []
    for col in predictors.columns:
        if not df[col].isnull().any():
            cols_with_no_nans.append(col)
    return cols_with_no_nans
```

```
num_cols = get_cols_with_no_nans(combined_data , 'num')
cat_cols = get_cols_with_no_nans(combined_data , 'no_num')
```

```
print ('Number of numerical columns with no nan values :',len(num_cols))
print ('Number of nun-numerical columns with no nan values :',len(cat_cols))
```

```
Number of numerical columns with no nan values : 101
Number of nun-numerical columns with no nan values : 26
```

```
import matplotlib.pyplot as plt
combined_data = combined_data[num_cols + cat_cols]
combined_data.hist(figsize = (20,15))
plt.show()
```



```
1 print(combined_data.head())
```

```
   state  fold  householdsize  racepctblack  racePctWhite  racePctAsian  \
0      8     1           0.33          0.02          0.90          0.12   
1     53     1           0.16          0.12          0.74          0.45   
2     24     1           0.42          0.49          0.56          0.17   
3     34     1           0.77          1.00          0.08          0.12   
4     42     1           0.55          0.02          0.95          0.09   

   racePctHisp  agePct12t21  agePct12t29  agePct16t24  ...  PctPolicAsian  \
0         0.17         0.34         0.47         0.29  ...            0.1   
1         0.07         0.26         0.59         0.35  ...              ?   
```

```
2        0.04         0.39         0.47         0.28  ...              ?
3        0.10         0.51         0.50         0.34  ...              ?
4        0.05         0.38         0.38         0.23  ...              ?

   PctPolicMinor  OfficAssgnDrugUnits  NumKindsDrugsSeiz  PolicAveOTWorked  \
0           0.07                 0.02               0.57              0.29
1              ?                    ?                  ?                 ?
2              ?                    ?                  ?                 ?
3              ?                    ?                  ?                 ?
4              ?                    ?                  ?                 ?

   PolicCars  PolicOperBudg  LemasPctPolicOnPatr  LemasGangUnitDeploy  \
0       0.06           0.04                  0.9                  0.5
1          ?              ?                    ?                    ?
2          ?              ?                    ?                    ?
3          ?              ?                    ?                    ?
4          ?              ?                    ?                    ?

   PolicBudgPerPop
0             0.14
1                ?
2                ?
3                ?
4                ?

[5 rows x 127 columns]
```

```
numeric_features = X.select_dtypes(include=['float64', 'int64']).columns
numeric_features = y.select_dtypes(include=['float64', 'int64']).columns
```

```
print("Shape of X_train_norm:", X_train_norm.shape)
print("Shape of X_test_norm:", X_test_norm.shape)
```

```
Shape of X_train_norm: (178, 13)
Shape of X_test_norm: (36, 13)
```

```
base_model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(1)  # Output layer with 1 neuron for regression
])
```

```
base_model.compile(optimizer='adam', loss='mean_squared_error')
```

```
base_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
base_model_loss = base_model.evaluate(X_test, y_test)
print("Base Model Loss:", base_model_loss)
```

```
Epoch 1/50
5/5 [==============================] - 1s 82ms/step - loss: 42881.6250 - val_loss: 34040.0586
Epoch 2/50
5/5 [==============================] - 0s 25ms/step - loss: 23412.1172 - val_loss: 17278.7422
Epoch 3/50
5/5 [==============================] - 0s 33ms/step - loss: 11260.1924 - val_loss: 6629.3672
Epoch 4/50
5/5 [==============================] - 0s 29ms/step - loss: 3829.4797 - val_loss: 1610.9077
Epoch 5/50
5/5 [==============================] - 0s 38ms/step - loss: 709.7225 - val_loss: 351.5442
Epoch 6/50
5/5 [==============================] - 0s 41ms/step - loss: 320.5483 - val_loss: 665.7840
Epoch 7/50
5/5 [==============================] - 0s 16ms/step - loss: 750.6833 - val_loss: 1009.4555
Epoch 8/50
5/5 [==============================] - 0s 17ms/step - loss: 937.4267 - val_loss: 905.8844
Epoch 9/50
5/5 [==============================] - 0s 19ms/step - loss: 736.7676 - val_loss: 581.3610
Epoch 10/50
5/5 [==============================] - 0s 31ms/step - loss: 422.6159 - val_loss: 354.5090
Epoch 11/50
5/5 [==============================] - 0s 40ms/step - loss: 240.7941 - val_loss: 300.8843
Epoch 12/50
5/5 [==============================] - 0s 34ms/step - loss: 197.7800 - val_loss: 333.8279
Epoch 13/50
5/5 [==============================] - 0s 70ms/step - loss: 214.2085 - val_loss: 357.6358
Epoch 14/50
5/5 [==============================] - 1s 140ms/step - loss: 221.9402 - val_loss: 343.2788
Epoch 15/50
5/5 [==============================] - 0s 33ms/step - loss: 203.7536 - val_loss: 301.2026
Epoch 16/50
5/5 [==============================] - 0s 21ms/step - loss: 179.9644 - val_loss: 268.5652
```

```
Epoch 17/50
5/5 [==============================] - 0s 35ms/step - loss: 170.9759 - val_loss: 252.7450
Epoch 18/50
5/5 [==============================] - 0s 60ms/step - loss: 166.7505 - val_loss: 244.6342
Epoch 19/50
5/5 [==============================] - 0s 32ms/step - loss: 163.5343 - val_loss: 237.1605
Epoch 20/50
5/5 [==============================] - 0s 39ms/step - loss: 156.7230 - val_loss: 230.5436
Epoch 21/50
5/5 [==============================] - 0s 79ms/step - loss: 150.0701 - val_loss: 224.9058
Epoch 22/50
5/5 [==============================] - 0s 39ms/step - loss: 144.9330 - val_loss: 220.1744
Epoch 23/50
5/5 [==============================] - 0s 50ms/step - loss: 139.9059 - val_loss: 213.9483
Epoch 24/50
5/5 [==============================] - 0s 38ms/step - loss: 135.5944 - val_loss: 206.8410
Epoch 25/50
5/5 [==============================] - 0s 53ms/step - loss: 131.3247 - val_loss: 200.4183
Epoch 26/50
5/5 [==============================] - 0s 65ms/step - loss: 127.5788 - val_loss: 193.3117
Epoch 27/50
5/5 [==============================] - 0s 34ms/step - loss: 123.7908 - val_loss: 188.1452
Epoch 28/50
5/5 [==============================] - 0s 38ms/step - loss: 120.0626 - val_loss: 181.6704
Epoch 29/50
5/5 [==============================] - 0s 63ms/step - loss: 116.2726 - val_loss: 177.4808
```

```python
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
improved_model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    Dense(64, activation='relu'),
    Dense(1)  # Output layer with 1 neuron for regression
])
```

```python
improved_model.compile(optimizer='adam', loss='mean_squared_error')
```

```python
improved_history = improved_model.fit(X_train_scaled, y_train, epochs=50, batch_size=32, validation_data=(X_test_scaled, y_test))

improved_model_loss = improved_model.evaluate(X_test_scaled, y_test)
print("Improved Model Loss:", improved_model_loss)
```