Name: Santiago, John Loyd C.
Course and Section: CPE 019 - CPE32S3
Date of Submission: April 10, 2024
Instructor: Engr. Roman Richard

1.Choose any dataset applicable to the classification problem, and also, choose any dataset applicable to the regression problem.

link of the google colab: https://colab.research.google.com/drive/1HPfAeZ3SJIl61gtyqjFWg3ZFs31eDG1T?usp=sharing

link of the dataset: https://www.kaggle.com/datasets/uom190346a/water-quality-and-potability?resource=download

2.Explain your datasets and the problem being addressed.

the problem is I'm trying to determine which water is safe for human consumption

3.For classification, do the following:

```
!pip install np_utils
```

```
Requirement already satisfied: np_utils in /usr/local/lib/python3.10/dist-packages (0.6.0)
Requirement already satisfied: numpy>=1.0 in /usr/local/lib/python3.10/dist-packages (from np_utils) (1.25.2)
```

```
!pip install scikeras
```

```
Requirement already satisfied: scikeras in /usr/local/lib/python3.10/dist-packages (0.12.0)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist-packages (from scikeras) (24.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (3.4.0)
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
from keras.models  import Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.optimizers import Adam, SGD, RMSprop
from scikeras.wrappers import KerasRegressor
from sklearn.utils import resample
from keras.models import Sequential
from keras.layers import Dense
from scikeras.wrappers import KerasClassifier
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder,MultiLabelBinarizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold

WP = pd.read_csv("./water_potability.csv")
```

```
seed = 7
np.random.seed(seed)
```

```
WP.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

```
WP.head(3276)
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0
...	...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	1
3272	7.808856	193.553212	17329.802160	8.061362	NaN	392.449580	19.903225	NaN	2.798243	1
3273	9.419510	175.762646	33155.578218	7.350233	NaN	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.869376	6.303357	NaN	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.177061	7.509306	NaN	327.459760	16.140368	78.698446	2.309149	1

3276 rows × 10 columns

Next steps: View recommended plots

I convert all NaN values to 0.

```
WP= WP.fillna(0)
```

```
WP.head(3276)
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	0.000000	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	0.000000	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	0.000000	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0
...	...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	1
3272	7.808856	193.553212	17329.802160	8.061362	0.000000	392.449580	19.903225	0.000000	2.798243	1
3273	9.419510	175.762646	33155.578218	7.350233	0.000000	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.869376	6.303357	0.000000	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.177061	7.509306	0.000000	327.459760	16.140368	78.698446	2.309149	1

3276 rows × 10 columns

Next steps: View recommended plots

```
count = WP['Potability'].value_counts()
print(count)
```

```
Potability
0    1998
1    1278
Name: count, dtype: int64
```

```
One = WP[WP['Potability'] == 1]
Zero = WP[WP['Potability'] == 0]
```

```
D1_resampled = resample(Zero, replace=False, n_samples=1278, random_state = 42)
water = pd.concat([D1_resampled, One])
```

```
count = water['Potability'].value_counts()
count
```

```
     Potability
     0    1278
     1    1278
     Name: count, dtype: int64
```

```
X = water.iloc[:, :-1].values
y = water["Potability"].values
```

```
X
```

```
     array([[6.26279887e+00, 2.06889748e+02, 3.14145258e+04, ...,
             1.59635398e+01, 7.30226053e+01, 4.01251756e+00],
            [7.80383322e+00, 2.23688111e+02, 3.73767930e+04, ...,
             1.66974084e+01, 7.47824342e+01, 2.90738727e+00],
            [8.69211532e+00, 1.44236358e+02, 2.55296280e+03, ...,
             1.39634211e+01, 4.23886613e+01, 2.28347516e+00],
            ...,
            [9.41951032e+00, 1.75762646e+02, 3.31555782e+04, ...,
             1.10390697e+01, 6.98454003e+01, 3.29887550e+00],
            [5.12676292e+00, 2.30603758e+02, 1.19838694e+04, ...,
             1.11689462e+01, 7.74882131e+01, 4.70865847e+00],
            [7.87467136e+00, 1.95102299e+02, 1.74041771e+04, ...,
             1.61403676e+01, 7.86984463e+01, 2.30914906e+00]])
```

```
y
```

```
     array([0, 0, 0, ..., 1, 1, 1])
```

```
normalizer = StandardScaler()
x_train_n = normalizer.fit_transform(X_train)
x_test_n = normalizer.transform(X_test)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=11111)
```

```
np.mean(y), np.mean(1-y)
```

```
     (0.5, 0.5)
```

Classification

- Create a base model

```
def baseline_model():
    model = Sequential()
    model.add(Dense(16, input_dim=9, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```
estimator = KerasClassifier(model=baseline_model, epochs=100, batch_size=200, verbose=0)
```

```
from sklearn.preprocessing import MultiLabelBinarizer
kfold = StratifiedKFold(n_splits=10, shuffle=True)
```

```
results = cross_val_score(estimator, x_train_n, y_train, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
     Baseline: 55.29% (2.99%)
```

Improve Version

```
def baseline_model():
    model = Sequential()
    model.add(Dense(70, input_dim=9, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))

    model.add(Dense(1, activation='sigmoid'))

  # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```
estimator = KerasClassifier(model=baseline_model, epochs=10, batch_size=10, verbose=0)
kfold = StratifiedKFold(n_splits=10, shuffle=True)
results = cross_val_score(estimator, x_train_n, y_train, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
     Baseline: 55.14% (3.68%)
```

4.For regression, do the following:

Regression

```
def baseline_model():
  model = Sequential()
  model.add(Dense(9, input_shape=(9,), kernel_initializer='normal', activation='relu'))
  model.add(Dense(1, kernel_initializer='normal'))
  model.compile(loss='mean_squared_error', optimizer='adam')
  return model
```

```
#evaluate model
estimator = KerasRegressor(model=baseline_model, epochs=100, batch_size=5, verbose=0)
kfold = KFold(n_splits=10)
results = cross_val_score(estimator, x_train_n, y_train, cv=kfold, scoring='neg_mean_squared_error')
print("Baseline: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
     Baseline: -0.25 (0.01) MSE
```

Standard

```
def baseline_model():
    # create model
  model = Sequential()
  model.add(Dense(9, input_shape=(9,), kernel_initializer='normal', activation='relu'))
  model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
  model.compile(loss='mean_squared_error', optimizer='adam')
  return model
# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=baseline_model, epochs=50, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, x_train_n, y_train, cv=kfold, scoring='neg_mean_squared_error')
print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
     Standardized: -0.25 (0.01) MSE
```

Standardized and larger

```
def larger_model():
    # create model
    model = Sequential()
    model.add(Dense(9, input_shape=(9,), kernel_initializer='normal', activation='relu'))
    model.add(Dense(3, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=larger_model, epochs=50, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, x_train_n, y_train, cv=kfold, scoring='neg_mean_squared_error')
print("Larger: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Larger: -0.25 (0.01) MSE
```

Wider

```
def wider_model():
 # create model
 model = Sequential()
 model.add(Dense(20, input_shape=(9,), kernel_initializer='normal', activation='relu'))
 model.add(Dense(1, kernel_initializer='normal'))
 # Compile model
 model.compile(loss='mean_squared_error', optimizer='adam')
 return model
# evaluate model with standardized dataset
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=wider_model, epochs=100, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, x_train_n, y_train, cv=kfold, scoring='neg_mean_squared_error')
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Wider: -0.25 (0.01) MSE
```

Conclusion

I concluded that I've learned the process of evaluating models using K-Fold cross-validation, which involves partitioning the data into 10 folds and shuffling it for robust assessment. Additionally, I've grasped the significance of standardizing the dataset and tuning the model by adjusting the number of layers and neurons. By incorporating more hidden layers, I've effectively enhanced the accuracy of my model, thereby optimizing its performance.