

Name: **Aishwarya Girish Menon**

Branch: **Computer Science and Engineering**

College: **REVA University**

QUESTION: 1

To predict if it will rain tomorrow in XYZ country using suitable ML approach.

PROGRAM: 1

```
import warnings
```

```
# import libraries for suppressing warnings
```

```
import pandas as pd
```

```
# import pandas libraries as pd for data analysis
```

```
import numpy as np
```

```
# import numpy libraries as np for handling multi-dimensional arrays
```

```
warnings.filterwarnings("ignore")
```

```
# this command is used to ignore or suppress any warnings
```

```
df = pd.read_csv(r"weatherAUS (1).csv")
```

```
# import weatherAUS dataset as dataframe using pd.read_csv
```

```
print('Size of weather data frame is :', df.shape)
```

Prints the size of the dataframe using df.shape() function. Here the dataset has 10705 rows and 23 columns.

df.head()

This command displays the top five records in the dataset

df.describe()

This command is used to observe different statistical patterns of the dataset columns and records such as standard deviation, mean, total count, minimum and maximum range etc.

col_names = df.columns

col_names

#initialize col_names variable and use df.columns to display all column names

df.count().sort_values()

This function is used to find number of values per column

df =

df.drop(columns=['Sunshine','Evaporation','Cloud3pm','Cloud9am','Date'],axis=1)

df.shape

the columns sunshine, evaporation, cloud3pm and cloud9am have least number of values while date column is irrelevant for prediction training. Hence these columns are removed using df.drop() function and the shape is displayed again.

```
df = df.dropna()
```

```
df.shape
```

```
# All columns containing NA values are removed using df.dropna() and  
shape is displayed.
```

```
from sklearn import preprocessing
```

```
# import preprocessing library from sklearn for cleaning data
```

```
numerical = [var for var in df.columns if df[var].dtype=='float64']
```

```
for col in numerical:
```

```
    df[col] = preprocessing.scale(df[col])
```

```
df.head()
```

```
# Standardize each numerical column by using for loop to traverse each  
record and preprocessing.scale() function, by creating a list to store all  
columns with numerical data. This is done to reduce unwanted errors in  
calculation.
```

```
from scipy import stats
```

```
#import stats from scipy library to handle outliers
```

```
z = np.abs(stats.zscore(df._get_numeric_data()))
```

```
print(z)
```

```
df= df[(z < 3).all(axis = 1 )]
```

```
print(df.shape)
```

```
# outliers can affect the accuracy of trained model. Stats.zcore() is used  
to calculate the Z score to remove outliers. Where ever the row of data  
has outlier beyond 3 standard deviations, the entire row is removed from  
dataset.
```

```
categorical = [var for var in df.columns if df[var].dtype=='object']  
print("Number of categorical variables: ", len(categorical))  
print(categorical)
```

This method is used to display all categorical data in the dataset and find the number of columns having categorical variables.

```
df['RainToday'].replace({'No': 0, 'Yes': 1},inplace = True)  
df['RainTomorrow'].replace({'No': 0, 'Yes': 1},inplace = True)
```

df.replace() is used to change all the yes and no values in the above two columns to 0 and 1 where 0 is No and 1 is Yes in order to make it easier for the model to train with the dataset. This method is called one hot encoding (converting categorical data to numerical data)

```
categorical_columns = ['Location','WindGustDir', 'WindDir3pm',  
'WindDir9am']
```

```
for col in categorical_columns:
```

```
    print(np.unique(df[col]))
```

```
df = pd.get_dummies(df, columns=categorical_columns)
```

the above lines of codes are used to convert the remaining categorical column variables to numerical data by using one hot encoding using pd.getdummies() function. np.unique helps to display all unique values in each categorical column.

```
from sklearn.model_selection import train_test_split
```

import train_test_split function from sklearn's model_selection library to segregate the dataset to training and testing dataset

```
X = df.loc[:,df.columns!='RainTomorrow']
```

```
y = df.RainTomorrow
```

```
# Divide the dataset such that X variable contains input values while y  
variable contains output values.
```

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
# segregate the dataset to training and testing parts where the testing  
dataset is of 20% size of total dataset.
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
%matplotlib inline
```

```
# import matplotlib and seaborn libraries to plot graphs for vsiaulization  
with KNN model.
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
# import KNeighboursClassifier function from sklearn for KNN  
classification.
```

```
k_list = list(range(1,50,2))
```

```
cv_scores = []
```

```
from sklearn.model_selection import cross_val_score
```

```
for k in k_list:
```

```
    knn = KNeighborsClassifier(n_neighbors=k)
```

```
    scores = cross_val_score(knn, x_train, y_train, cv=10,  
scoring='accuracy')
```

```
    cv_scores.append(scores.mean())
```

```
MSE = [1 - x for x in cv_scores]
```

initially the cv scores are obtained to check the skill of ML model for each parameter. 10-fold cross validation is performed on the dataset using KNeighborsClassifier to find the K value having least misclassification error by tracking the score using cross_val_score(). The MSE score for each number of K neighbours is tracked.

```
plt.figure()
```

```
plt.figure(figsize=(15,10))
```

```
plt.title('The optimal number of neighbors', fontsize=20,  
fontweight='bold')
```

```
plt.xlabel('Number of Neighbors K', fontsize=15)
```

```
plt.ylabel('Misclassification Error', fontsize=15)
```

```
sns.set_style("whitegrid")
```

```
plt.plot(k_list, MSE)
```

```
plt.show()
```

A matplotlib plot is used to plot the misclassification error for each value of K (number of neighbours) by using plt.plot(). plt.xlabel and plt.ylabel help in naming the axes while plt.title provides title of graph. plt.figure() sets the size of the plotted graph and plt.show() displays the graph.

```
best_k = k_list[MSE.index(min(MSE))]
```

```
print("The optimal number of neighbors is %d." % best_k)
```

the above function is used to find the optimal value of K based on the MSE scores.

```
from sklearn.neighbors import KNeighborsClassifier
```

```
k = 23
```

```
knn = KNeighborsClassifier(n_neighbors=k)
```

```
knn.fit(x_train, y_train)
```

```
preds = knn.predict(x_test)
```

The KNN model is now trained using K value as 23 which is the optimal value using the KNeighborsClassifier() function and the model training is initiated using knn.fit() function here where training dataset is passed as input. The knn.predict() function is used to test the trained model with testing dataset.

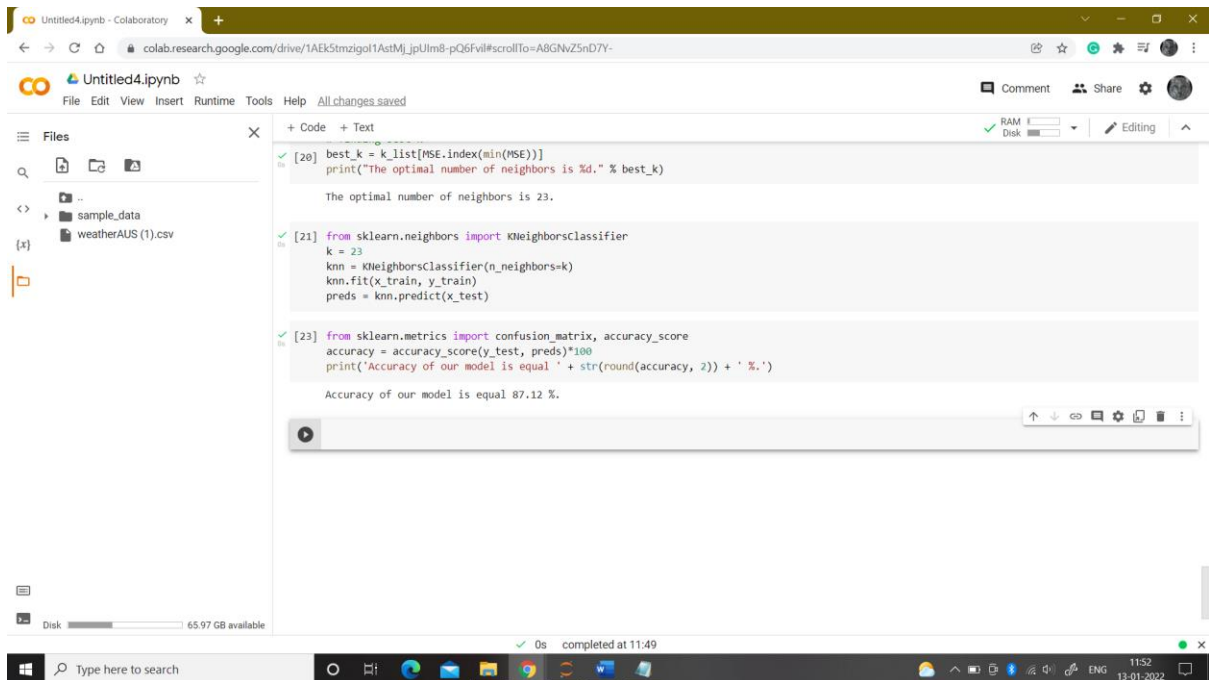
```
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
accuracy = accuracy_score(y_test, preds)*100
```

```
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + '%.')
```

The accuracy score of the trained model is calculated by using the above formula that uses the prediction results and testing dataset with the help of accuracy_score () function.

RESULT:



```
[20] best_k = k_list[MSE.index(min(MSE))]
      print("The optimal number of neighbors is %d." % best_k)

      The optimal number of neighbors is 23.

[21] from sklearn.neighbors import KNeighborsClassifier
      k = 23
      knn = KNeighborsClassifier(n_neighbors=k)
      knn.fit(x_train, y_train)
      preds = knn.predict(x_test)

[23] from sklearn.metrics import confusion_matrix, accuracy_score
      accuracy = accuracy_score(y_test, preds)*100
      print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')

      Accuracy of our model is equal 87.12 %.
```

The final accuracy score of the model was 87.12% after performing model training using KNN classifier which underwent hyperparameter tuning to determine optimal K value. The resultant model predicts whether it will rain or not in the given city.

QUESTION: 2

Train a deep learning classification model for classes in MNIST Handwriting dataset.

PROGRAM: 2

import tensorflow as tf

tensorflow library is imported as tf for numerical computing and machine learning tasks


```
from matplotlib import pyplot as plt
```

```
# pyplot is imported from matplotlib as plt for 2D graphics visualization of  
dataset or images.
```

```
import numpy as np
```

```
# importing numpy library as np for working with the arrays
```

```
from tensorflow.keras.datasets import mnist
```

```
# the MNIST dataset is imported from the sample datasets provided by  
Keras library from tensorflow library and interfaces
```

```
(training_images, training_labels), (test_images, test_labels) =  
mnist.load_data()
```

```
# divide the dataset into training section (with training images and its  
labels) and testing section (with testing images and its labels) from the  
MNIST dataset.
```

```
for i in range(9):
```

```
    plt.subplot(330 + 1 + i)
```

```
    plt.imshow(training_images[i])
```

```
# Using for loop, traverse through the training section to display the first  
9 images by defining subplot with plt.subplot() and plotting the raw pixel  
data in 2D using plt.imshow() to display the output.
```

```
print(training_images.shape)
```

```
print(training_images[0])
```

```
# the training_images.shape command is used to check how each  
image looks and its pixel values in terms of width and height in matrix  
form. A sample image is used for this purpose at index 0 of the training  
section. The pixel values range from 0 to 255
```

```
training_images = training_images / 255.0
```

```
test_images = test_images / 255.0
```

```
# Normalize the data to a scale of 0 to 1 by dividing training and testing  
dataset by 255 to simplify the dataset for faster computation
```

```
model =
```

```
tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28,  
28)),
```

```
tf.keras.layers.Dense(128, activation='relu'),
```

```
tf.keras.layers.Dense(10,  
activation=tf.nn.softmax)])
```

```
# a variable model is initialized and a Sequential () function, from the  
keras model's library is used for creating a layer of networks in a  
sequence. Each layer is given 28x28 neurons.
```

Flatten function is used to convert each 2D image structure to 1D to send as input to each row of neurons. The Dense function is used to create next layer where we connect each neuron in previous layer to the neuron in the next layer with the activation function used being ReLu and the hidden layer having 128 neurons.

The next dense function is used to create output layer with the number of neurons in output layer is equal to 10 as there are 10 classes from 0 to 9. The activation function used is softmax to output the maximum probability which is selected as output.

```
model.compile(optimizer = 'adam',
```

```
loss = 'sparse_categorical_crossentropy',
```

```
metrics=['accuracy'])
```

```
# The optimization function and loss function is initialized where the  
optimizer used is adam (modified version of gradient descent algorithm).  
Cross-entropy is used to measure loss score using loss variable to print
```

during training. The metrics is set to print accuracy score for every epoch during training with the loss score. The optimizer is used to adjust the weights and learning rate of the model for accuracy improvement.

model.fit(training_images, training_labels, epochs=5)

The model is the trained using model.fit() function which takes the training images and labels as input with the number of epochs set to 5. The number of epochs determines the number of complete passes through training dataset. The accuracy score is displayed at the end for training dataset.

print(model.evaluate(test_images, test_labels))

model.evaluate() function is used to test the accuracy score for the testing section dataset by passing testing images and labels as parameter.

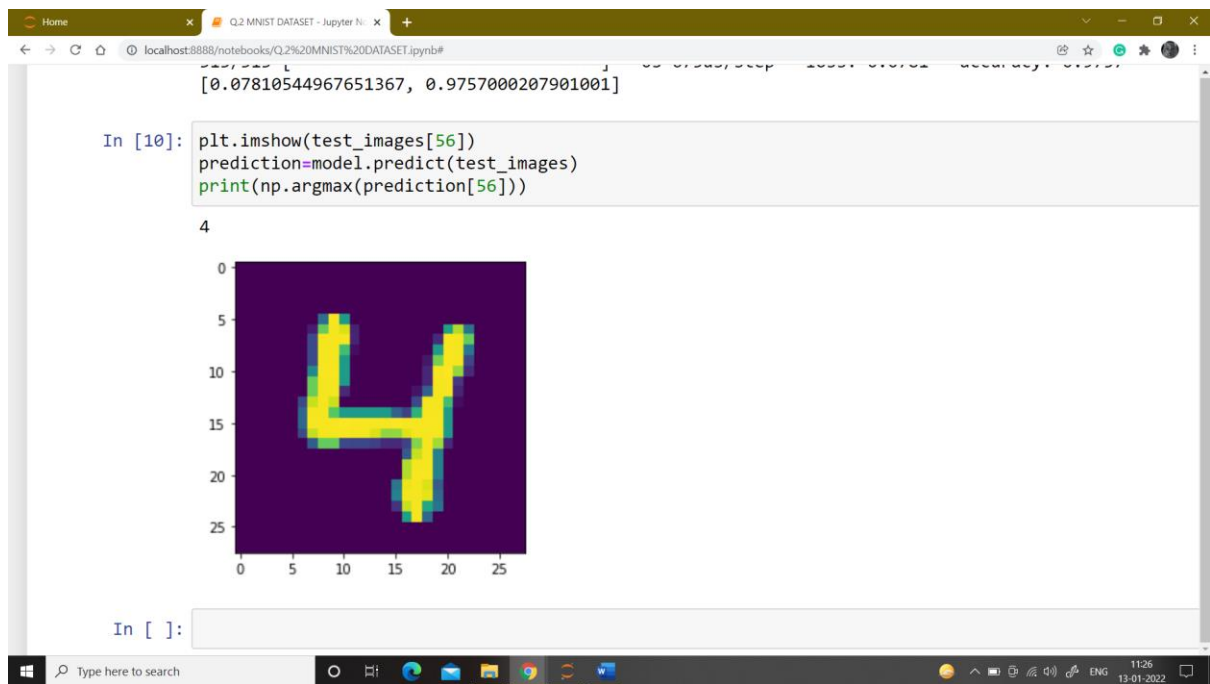
plt.imshow(test_images[56])

prediction=model.predict(test_images)

print(np.argmax(prediction[56]))

The np.argmax() function is used to check which neuron has the maximum probability. The plt.imshow() function is used to display the sample image for which prediction is to be made. The prediction variable will contain the prediction label obtained using model.predict() function by passing the test images dataset.

RESULT:



The trained model was testing by passing the 57th image in the dataset as input to predict the digit in the image. The image was predicted to be digit number 4 as shown above. The accuracy scores were as follows:

Training – 98.61%

Testing – 97.58%