Practical: 1

Aim: Prepare a study on environment set up for Tensor Flow and Google Colab. Also implement

the basic python commands related to Machine Learning.

Tensor Flow:

Title: Environmental Setup for TensorFlow: A Comprehensive Study

Abstract: TensorFlow is a popular open-source machine learning framework developed by Google that has gained widespread adoption in the field of deep learning. Setting up the appropriate environment for TensorFlow is a crucial step in developing and running machine learning models efficiently. This study provides a comprehensive guide to setting up the environment for TensorFlow, covering installation, configuration, and best practices to ensure a smooth and productive development process.

1. Introduction:

- Brief overview of TensorFlow and its significance in machine learning.

2. System Requirements:

- Hardware prerequisites (CPU, GPU, RAM, and storage).
- Supported operating systems (Linux, Windows, macOS).

3. Installation:

- Overview of installation methods (pip, Anaconda, Docker).
- Choosing the appropriate TensorFlow version (CPU or GPU).
- Step-by-step installation instructions for each method.
- Benefits of using virtual environments.
- Creating and managing Python virtual environments.
- Installing TensorFlow within a virtual environment.

4. GPU Setup (if applicable):

5. Additional Libraries and Dependencies:

- Installing andmanaging required Python packages (NumPy, Pandas,Matplotlib, etc.).

- Version compatibility considerations.
- Using package managers (pip or conda) for dependency management.

6. TensorFlow Configuration:

- Configuring TensorFlow for optimal performance and compatibility.
- TensorFlow's built-in configurations.
- Customizing configurations for specific use cases.

7. IDEs and Development Tools:

- Overview of popular integrated development environments (IDEs) for TensorFlow.
- Setting up IDEs (e.g., TensorFlow in Jupyter Notebook, PyCharm, VisualStudio Code).
- Debugging and profiling tools.

8. Data Management:

- Data preparation and preprocessing.
- Handling datasets using TensorFlow Datasets and TensorFlow Data Services.
- Integration with popular data manipulation libraries (e.g., TensorFlow Data Validation).

9. Best Practices:

- Ensuring code portability across different environments.
- Using version control (e.g., Git) for code management.
- Managing project dependencies efficiently.
- Creating a reproducible environment (e.g., using requirements.txt orenvironment.yml).

10. Troubleshooting and Common Issues:

- Identifying and resolving installation and configuration problems.
- Community resources and forums for assistance.

11. Conclusion:

- Recap of the importance of a well-configured environment for TensorFlow.
- Key takeaways from the study.

12. References:

- Citations and links to official TensorFlow documentation and relevant resources.

By following the guidelines presented in this comprehensive study, developers and researchers can establish a robust and efficient environment for TensorFlow, facilitating the development and deployment of machine learning models.

Google Colab:-

Title: Setting Up the Environment for Google Colab: A Comprehensive Study

Abstract: Google Colab is a cloud-based, interactive computing environment that provides free access to powerful GPU and TPU resources, making it an attractive platform for machinelearning and data science tasks. This study offers a thorough guide to setting up the environment in Google Colab, covering topics such as connecting to a runtime, installing libraries, managing data, and maximizing productivity for a seamless development experience.

1. Introduction:

- Overview of Google Colab and its popularity in the data science and machine learning communities.
- The significance of a well-configured environment for efficient Colab usage.

2. Accessing Google Colab:

- Creating a Google account (if not already available).
- Accessing Google Colab via a web browser.
- Setting up a Google Drive account for easy data storage and access.

3. Colab Runtime Environment: 3.1. Runtime Types:

- Understanding the different runtime types (CPU, GPU, TPU).
- Selecting an appropriate runtime for your tasks.

4. Installing Libraries and Dependencies:

- Using package managers like !pip and !apt to install Python libraries.
- Installing popular data science and machine learning libraries (e.g., NumPy, Pandas, TensorFlow, PyTorch).

5. Data Management: 5.1. Uploading Data:

- Uploading data files to the Colab environment.
- Accessing data stored in Google Drive.

5.2. Using External Data Sources:

- Accessing datasets from popular sources like Kaggle, GitHub, and Google Drive.
- Mounting Google Drive for easy access to data.

6. Version Control:

- Setting up and using Git for version control within Colab.
- Collaborative coding and sharing notebooks through GitHub integration.

7. Maximizing Productivity: 7.1. Magic Commands:

- Using Colab's built-in magic commands (%).
- Examples of magic commands for efficiency (e.g., %cd, %time, %load).

7.2. Keyboard Shortcuts:

- Essential keyboard shortcuts for efficient coding.
- Customizing keyboard shortcuts for personal preferences.

7.3. GPU/TPU Utilization:

- Monitoring and optimizing GPU/TPU usage.
- Guidelines for efficient memory management.

8. Troubleshooting:

- Common issues and error messages in Colab
  - Tips for resolving runtime and package installation problems.
- Security and Privacy:
  - Best practices for handling sensitive data in a cloud-based environment.
  - Understanding the security implications of Google Colab.
- Conclusion:
  - Recap of the importance of a well-configured environment for Google Colab.
  - Key takeaways from the study.
- References:

○ Citations and links to official Google Colab documentation and relevant resources.

By following the guidelines outlined in this comprehensive study, users can effectively set upand optimize their Google Colab environment for a wide range of data science and machine learning tasks, enhancing productivity and performance.

Implementing the basic python command which used in machine learning

1) String to Integer (int):

```
str_num = "42"
int_num = int(str_num)
print(int_num)  # Output: 42
```

```
42
```

2) String to Float (float):

```
str_float = "3.14"
float_num = float(str_float)
print(float_num)  # Output: 3.14
```

```
3.14
```

3) Integer to String:

```
int_num = 42
str_num = str(int_num)
print(str_num)
```

```
42
```

4) Float to String:

```
float_num = 3.14
str_float = str(float_num)
print(str_float)
```

```
3.14
```

5) List to Tuple:

```
my_list = [1, 2, 3]
my_tuple = tuple(my_list)
print(my_tuple)
```

```
(1, 2, 3)
```

6) Tuple to List:

```
my_tuple = (1, 2, 3)
my_list = list(my_tuple)
print(my_list)
```

```
[1, 2, 3]
```

7) String to List of Characters:

```
my_string = "Hello"
char_list = list(my_string)
print(char_list)
```

```
['H', 'e', 'l', 'l', 'o']
```

8) List of Integers to a String (Joining):

```
num_list = [1, 2, 3, 4, 5]
num_str = ''.join(map(str, num_list))
print(num_str)
```

```
12345
```

9) String to List of Words:

```
sentence = "This is a sample sentence"
word_list = sentence.split()
print(word_list)
```

```
['This', 'is', 'a', 'sample', 'sentence']
```

10) List of Strings to a Single String (Joining):

```
word_list = ['This', 'is', 'a', 'list']
sentence = ' '.join(word_list)
print(sentence)
```

```
This is a list
```

String:-

```
my_string = "Hello, Python!"
print(my_string)
```

```
Hello, Python!
```

Sliceing:-

```
my_string = "Hello, World!"
sliced_str = my_string[7:12]  # Extract "World"
print(sliced_str)
```

```
World
```

If else statement:-

```
num = int(input("Enter a number: "))

# Check if the number is even or odd
if num % 2 == 0:
    print(f"{num} is even.")
else:
    print(f"{num} is odd.")
```

```
Enter a number: 2
2 is even.
```

For loop statement:-

```
for i in range(1, 6):
    print(i)
```

```
1
2
3
4
5
```

While loop:

```
count = 5
while count > 0:
    print(count)
    count -= 1
```

```
5
4
3
2
1
```

Nestedloop:-

```
for i in range(3):
    for j in range(3):
        print(f"({i}, {j})")
```

```
(0, 0)
(0, 1)
(0, 2)
(1, 0)
(1, 1)
(1, 2)
(2, 0)
(2, 1)
(2, 2)
```

Python Data structure:

List: -

```
fruits = ["apple", "banana", "cherry"]
print(fruits[0])
fruits.append("orange")
fruits.remove("banana")
```

```
apple
```

Tuple: -

```
coordinates = (3, 4)
x, y = coordinates
print(x,y)
```

```
3 4
```

Set: -

```
colors = {"red", "green", "blue"}
colors.add("yellow")
colors.remove("green")
print(colors)
```

```
{'yellow', 'red', 'blue'}
```

Dictionary: -

```
person = {"name": "Alice", "age": 30}
print(person["name"])
person["city"] = "New York"
```

```
Alice
```

Range() function:-

```
numbers = list(range(1, 6))
print(numbers)
```

```
[1, 2, 3, 4, 5]
```

Formatting the string: -

```
name = "Alice"
age = 30
formatted_string = f"My name is {name} and I am {age} years old."
print(formatted_string)
```

```
My name is Alice and I am 30 years old.
```

Practical: 2

Aim: Implement the following data manipulation commands/functions:

- a) **Loading a CSV file.**
- b) **Save data from CSV file to Dataframe.**
- c) **Calculation of mean, median, variance, quartiles and inter-quartile range.**

Code:

a) Loading a CSV file:

```
from google.colab import drive

drive.mount('/content/drive')

import pandas as pd

df = pd.read_csv('/content/drive/My Drive/diabetes.csv')

print(df)

df.head(10)
```

Output:

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

```
Single column value using dataframe[]
0      148
1       85
2      183
3       89
4      137
      ...
763    101
764    122
765    121
766    126
767     93
Name: Glucose, Length: 768, dtype: int64
mean : 3.8450520833333335
```

b) Save data from CSV file to Dataframe:

print("Single column value using dataframe[]")

print(df['Glucose'])

Output:

```
Single column value using dataframe[]
0      148
1       85
2      183
3       89
4      137
      ...
763    101
764    122
765    121
766    126
767     93
Name: Glucose, Length: 768, dtype: int64
```

c) Calculation of mean, median, variance, quartiles and inter-quartile range:

mean1 = df['Age'].mean()

print('mean : ' + str(mean1))

Output:

```
mean : 33.240885416666664
```

median = df['Insulin'].median()

print('median : ' + str(median))

Output:

```
median : 30.5
```

mode = df['Age'].mode()

print('mode : ' + str(mode))

Output:

```
mode : 0    22
Name: Age, dtype: int64
```

variance = df['Age'].var()

print('variance : ' + str(variance))

Output:

```
variance : 138.30304589037377
```

average = df['BloodPressure'].mean()

print('average : ' + str(average))

Output:

```
average : 69.10546875
```

variance = df['BloodPressure'].var()

print('variance : ' + str(variance))

Output:

```
variance : 374.6472712271838
```

standard deviation = df['Pregnancies'].std()

print('standard deviation : ' + str(standard deviation))

Output:

```
standard deviation : 3.3695780626988694
```

import numpy as np

column_name = 'Age'

data = df[column_name]

q1 = np.percentile(data, 25)

q3 = np.percentile(data, 75)

iqr = q3 - q1

print(f'1st Quartile (Q1): {q1}')

print(f'3rd Quartile (Q3): {q3}')

print(f'Inter-Quartile Range (IQR): {iqr}')

Output:

```
1st Quartile (Q1): 24.0
3rd Quartile (Q3): 41.0
Inter-Quartile Range (IQR): 17.0
```

Practical: 3

Aim: Write a program to implement the naïve Bayesian classifier for Cancer data set stored as a .CSV file. Compute the accuracy of the classifier.

Code:

a) Importing Libraries.

```
In [2]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
```

b) Reading cancer dataset.

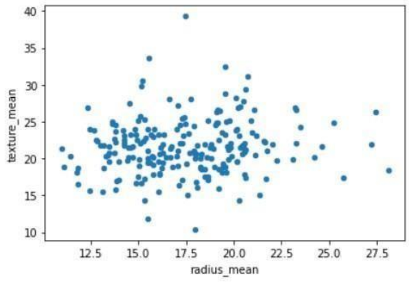
```
In [4]: dataset = pd.read_csv("data.csv")
        dataset.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 569 entries, 0 to 568
        Data columns (total 33 columns):
         #   Column                   Non-Null Count  Dtype
        ---  ------                   --------------  -----
         0   id                       569 non-null    int64
         1   diagnosis                569 non-null    object
         2   radius_mean              569 non-null    float64
         3   texture_mean             569 non-null    float64
         4   perimeter_mean           569 non-null    float64
         5   area_mean                569 non-null    float64
         6   smoothness_mean          569 non-null    float64
         7   compactness_mean         569 non-null    float64
         8   concavity_mean           569 non-null    float64
         9   concave points_mean      569 non-null    float64
         10  symmetry_mean            569 non-null    float64
         11  fractal_dimension_mean   569 non-null    float64
         12  radius_se                569 non-null    float64
         13  texture_se               569 non-null    float64
         14  perimeter_se             569 non-null    float64
         15  area_se                  569 non-null    float64
         16  smoothness_se            569 non-null    float64
         17  compactness_se           569 non-null    float64
         18  concavity_se             569 non-null    float64
         19  concave points_se        569 non-null    float64
         20  symmetry_se              569 non-null    float64
         21  fractal_dimension_se     569 non-null    float64
         22  radius_worst             569 non-null    float64
         23  texture_worst            569 non-null    float64
         24  perimeter_worst          569 non-null    float64
         25  area_worst               569 non-null    float64
         26  smoothness_worst         569 non-null    float64
         27  compactness_worst        569 non-null    float64
```

c) Processing dataset.

```
In [5]: dataset = dataset.drop(["id"], axis = 1)
        dataset = dataset.drop(["Unnamed: 32"], axis = 1)
```

```
In [6]: M = dataset[dataset.diagnosis == "M"]
        B = dataset[dataset.diagnosis == "B"]
```

```
In [15]: #plt.title("Malignant vs Benign Tumor")
         #plt.xlabel("Radius Mean")
         #plt.ylabel("Texture Mean")
         #plt.scatter(M.radius_mean, M.texture_mean, color = "red", label = "Malignant", alpha = 0.3)
         #plt.scatter(B.radius_mean, B.texture_mean, color = "lime", label = "Benign", alpha = 0.3)
         #plt.legend()
         #plt.show()
         g1 = dataset.loc[dataset.diagnosis=='M',:]
         # dataframe.plot.scatter() method
         g1.plot.scatter('radius_mean','texture_mean');
```




```
In [ ]: dataset.diagnosis = [1 if i== "M" else 0 for i in dataset.diagnosis]
```

```
In [ ]: x = dataset.drop(["diagnosis"], axis = 1)
        y = dataset.diagnosis.values
```

```
In [ ]: from sklearn.model_selection import train_test_split
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 42)
```

d) Applying the naïve Bayesian classifier.

```
In [ ]: from sklearn.naive_bayes import GaussianNB
        nb = GaussianNB()
        nb.fit(x_train, y_train)
        GaussianNB()
        print("Naive Bayes score: ",nb.score(x_test, y_test))

        Naive Bayes score:  0.9415204678362573
```

Practical: 4

Aim: Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Compute the accuracy of the classifier.

Code:

a) Importing and reading dataset:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

```
#read data from csv
data = pd.read_csv('iris.csv')
data.head()
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

```
print(data.keys())
```

```
Index(['sepal.length', 'sepal.width', 'petal.length', 'petal.width',
       'variety'],
      dtype='object')
```

b) Segregating predictors and target variable:

```
predictors = data.iloc[:,0:4]
target = data.iloc[:,4]
```

```
[ ] predictors_train, predictors_test, target_train, target_test = train_test_split(predictors,target,test_size=0.3,random_state=123)
```

c) Applying KNN Algorithm:

```
[ ] predictors_train, predictors_test, target_train, target_test = train_test_split(predictors,target,test_size=0.3,random_state=123)
```

```
#instantiate the model with 3 neighbors
nn = KNeighborsClassifier(n_neighbors=3)
```

```
[ ] #train model/classifier with input dataset
model = nn.fit(predictors_train,target_train)
```

d) Result:

```
[ ]  result = nn.predict([[5, 3, 2, 1],])

     print(result)
```

```
#Check prediction accuracy
nn.score(predictors_test, target_test)
```

```
0.9555555555555556
```

Practical: 5

Aim: Write a program to demonstrate the working of the decision tree algorithm. Use Cancer data set for building the decision tree and apply this knowledge to classify a new sample.

Code:

a) Importing and reading dataset:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
```

```
#read data from csv
data = pd.read_csv('Breast-Cancer-Wisconsin-Diagnostic-DataSet.csv')
data.head()
```

b) Segregating predictors and target variable:

```
predictors = data.iloc[:,0:31]
target = data.iloc[:,31]
```

```
predictors_train, predictors_test, target_train, target_test = train_test_split(predictors, target, test_size=0.3, random_state=123)
```

c) Applying Decision Tree algorithm:

```
#Decision Tree Classifier
dtree_entropy = DecisionTreeClassifier(criterion="entropy", random_state=100, max_depth=3, min_samples_leaf=5)
```

```
#train model/classifier with input dataset
model = dtree_entropy.fit(predictors_train, target_train)
```

d) Result:

```
prediction = dtree_entropy.predict(predictors_test)
```

```
accuracy_score(target_test, prediction, normalize=True)
```

```
0.9532163742690059
```

Practical: 6

Aim: Write a program to implement Random Forest Algorithm to classify Cancer data set.

Code:

a) Importing and reading dataset:

```
from google.colab import files

data = files.upload()
```

```
Choose Files Breast-Can...DataSet.csv
• Breast-Cancer-Wisconsin-Diagnostic-DataSet.csv(text/csv) - 125141 bytes, last modified: 11/5/2023 - 100% done
Saving Breast-Cancer-Wisconsin-Diagnostic-DataSet.csv to Breast-Cancer-Wisconsin-Diagnostic-DataSet (1).csv
```

b) Implementing Random Forest Algorithm:

```
# Import the required libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Random Forest classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
rf.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = rf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

c) Result:

```
Accuracy: 0.9649122807017544
```

Practical: 7

Aim: Write a program to implement Random Forest Algorithm to classify Cancer data set. Write a program to implement Support Vector Machine Algorithm to classify Cancer data set.

Code:

a) **Importing and reading dataset:**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import svm
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix, classification_report
```

```
[ ] #read data from csv
    data = pd.read_csv('Breast-Cancer-Wisconsin-Diagnostic-DataSet.csv')
    data.head()
```

b) **Segregating predictors and target variable:**

```
[23] predictors = data.iloc[:,0:31] #seggregating predictors variable
     target = data.iloc[:,31] #seggregating target variable
```

```
[24] predictors_train, predictors_test, target_train, target_test = train_test_split(predictors, target, test_size=0.3, random_state=123)
```

c) **Applying Random Forest Algorithm:**

```
#train model/classifier with input dataset
#model = svm.fit(predictors_train, target_train)
svm = svm.LinearSVC(C=100)
scaler = MinMaxScaler()
scaler.fit(predictors_train)
predictors_train = scaler.transform(predictors_train)
predictors_test = scaler.transform(predictors_test)
svm.fit(predictors_train,target_train)
```

```
[27] prediction = svm.predict(predictors_test)
     prediction
```

d) **Result:**

```
[28] #Check prediction accuracy
     accuracy_score(target_test, prediction, normalize=True)

     0.6023391812865497
```

```
confusion = confusion_matrix(target_test, prediction)
report = classification_report(target_test, prediction, target_names = ['malignant', 'benign'])
print('Confusion matrix: \n{}'.format(confusion))
print(report)
```

```
Confusion matrix:
[[103   0]
 [ 68   0]]
              precision    recall  f1-score   support

   malignant       0.60      1.00      0.75       103
      benign       0.00      0.00      0.00        68

    accuracy                           0.60       171
   macro avg       0.30      0.50      0.38       171
weighted avg       0.36      0.60      0.45       171
```

Practical: 8

Aim: Write a program to implement K-means Clustering.

Code:

a) **Importing and reading dataset:**

```
import numpy as np
import pandas as pd
import sklearn
```

```
from sklearn.datasets import load_digits
from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, homogeneity_score, completeness_score
from scipy.stats import mode
```

b) **Applying K-means Clustering:**

```
digits = load_digits()
digits_data =  digits.data/255
```

```
kmeans = KMeans(n_clusters=10, random_state=0)
digits_kmeans = kmeans.fit_predict(digits_data)
```

c) **Result:**

```
# get_cluster_accuracy(digits.target, digits_kmeans, 10)
labels = np.zeros_like(digits_kmeans)
for i in range(10):
    mask = (digits_kmeans == i)
    labels[mask] = mode(digits.target[mask])[0]
print("Accuracy {0} \n Homogeneity {1} \n Completeness {2}".format(
    accuracy_score(digits.target, labels), homogeneity_score(digits.target, labels),
completeness_score(digits.target, labels)))
```

```
Accuracy 0.7935447968836951
 Homogeneity 0.7423769268336259
 Completeness 0.7514312243853245
```

Practical: 9

Aim: Write a program to implement Apriori algorithm for association rule learning.

Code:

a) **Importing and reading dataset:**

```
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori, association_rules
```

```
df = pd.read_csv('GroceryStoreDataSet.csv', names = ['products'], sep = ',')
df.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `sh
  and should_run_async(code)
```

	products
0	MILK,BREAD,BISCUIT
1	BREAD,MILK,BISCUIT,CORNFLAKES
2	BREAD,TEA,BOURNVITA
3	JAM,MAGGI,BREAD,MILK
4	MAGGI,TEA,BISCUIT

b) **Applying Apriori algorithm:**

```
data = list(df["products"].apply(lambda x:x.split(",") ))
data
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: Deprec
  and should_run_async(code)
[['MILK', 'BREAD', 'BISCUIT'],
 ['BREAD', 'MILK', 'BISCUIT', 'CORNFLAKES'],
 ['BREAD', 'TEA', 'BOURNVITA'],
 ['JAM', 'MAGGI', 'BREAD', 'MILK'],
 ['MAGGI', 'TEA', 'BISCUIT'],
 ['BREAD', 'TEA', 'BOURNVITA'],
 ['MAGGI', 'TEA', 'CORNFLAKES'],
 ['MAGGI', 'BREAD', 'TEA', 'BISCUIT'],
 ['JAM', 'MAGGI', 'BREAD', 'TEA'],
 ['BREAD', 'MILK'],
 ['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],
 ['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],
 ['COFFEE', 'SUGER', 'BOURNVITA'],
 ['BREAD', 'COFFEE', 'COCK'],
 ['BREAD', 'SUGER', 'BISCUIT'],
 ['COFFEE', 'SUGER', 'CORNFLAKES'],
 ['BREAD', 'SUGER', 'BOURNVITA'],
 ['BREAD', 'COFFEE', 'SUGER'],
 ['BREAD', 'COFFEE', 'SUGER'],
 ['TEA', 'MILK', 'COFFEE', 'CORNFLAKES']]
```

```
#Let's transform the list, with one-hot encoding
from mlxtend.preprocessing import TransactionEncoder
a = TransactionEncoder()
a_data = a.fit(data).transform(data)
df = pd.DataFrame(a_data,columns=a.columns_)
df = df.replace(False,0)
df
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call
  and should_run_async(code)
```

	BISCUIT	BOURNVITA	BREAD	COCK	COFFEE	CORNFLAKES	JAM	MAGGI	MILK	SUGER	TEA
0	True	0	True	0	0	0	0	0	True	0	0
1	True	0	True	0	0	True	0	0	True	0	0
2	0	True	True	0	0	0	0	0	0	0	True
3	0	0	True	0	0	0	True	True	True	0	0
4	True	0	0	0	0	0	0	True	0	0	True
5	0	True	True	0	0	0	0	0	0	0	True
6	0	0	0	0	0	True	0	True	0	0	True
7	True	0	True	0	0	0	0	True	0	0	True
8	0	0	True	0	0	0	True	True	0	0	True
9	0	0	True	0	0	0	0	0	True	0	0
10	True	0	0	True	True	True	0	0	0	0	0
11	True	0	0	True	True	True	0	0	0	0	0
12	0	True	0	0	True	0	0	0	0	True	0
13	0	0	True	True	True	0	0	0	0	0	0
14	True	0	True	0	0	0	0	0	0	True	0
15	0	0	0	0	True	True	0	0	0	True	0
16	0	True	True	0	0	0	0	0	0	True	0
17	0	0	True	0	True	0	0	0	0	True	0
18	0	0	True	0	True	0	0	0	0	True	0
19	0	0	0	0	True	True	0	0	True	0	True

c) Calculating Support Values:

```
#set a threshold value for the support value and calculate the support value.
df = apriori(df, min_support = 0.2, use_colnames = True, verbose = 1)
df
```

```
Processing 42 combinations | Sampling itemset size 3
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` wi
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataF
  warnings.warn(
```

	support	itemsets
0	0.35	(BISCUIT)
1	0.2	(BOURNVITA)
2	0.65	(BREAD)
3	0.4	(COFFEE)
4	0.3	(CORNFLAKES)
5	0.25	(MAGGI)
6	0.25	(MILK)
7	0.3	(SUGER)
8	0.35	(TEA)
9	0.2	(BISCUIT, BREAD)
10	0.2	(BREAD, MILK)
11	0.2	(BREAD, SUGER)
12	0.2	(BREAD, TEA)
13	0.2	(COFFEE, CORNFLAKES)
14	0.2	(SUGER, COFFEE)
15	0.2	(TEA, MAGGI)

d) Association rule:

```
#Let's view our interpretation values using the Associan rule function.
df_ar = association_rules(df, metric = "confidence", min_threshold = 0.6)
df_ar
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically
  and should_run_async(code)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(MILK)	(BREAD)	0.25	0.65	0.2	0.800000	1.230769	0.0375	1.75	0.250000
1	(SUGER)	(BREAD)	0.30	0.65	0.2	0.666667	1.025641	0.0050	1.05	0.035714
2	(CORNFLAKES)	(COFFEE)	0.30	0.40	0.2	0.666667	1.666667	0.0800	1.80	0.571429
3	(SUGER)	(COFFEE)	0.30	0.40	0.2	0.666667	1.666667	0.0800	1.80	0.571429
4	(MAGGI)	(TEA)	0.25	0.35	0.2	0.800000	2.285714	0.1125	3.25	0.750000

Practical: 10

Aim: Write a program for prediction using Linear Regression on Boston Housing Dataset.

Code:

a) Importing and reading dataset:

```
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics
import pandas as pd
```

```
# load the boston dataset
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
X = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
y = raw_df.values[1::2, 2]
```

b) Splitting data set:

```
# splitting X and y into training and testing sets
X_train, X_test,\
    y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=1)
```

```
# create linear regression object
reg = linear_model.LinearRegression()

# train the model using the training sets
reg.fit(X_train, y_train)
```

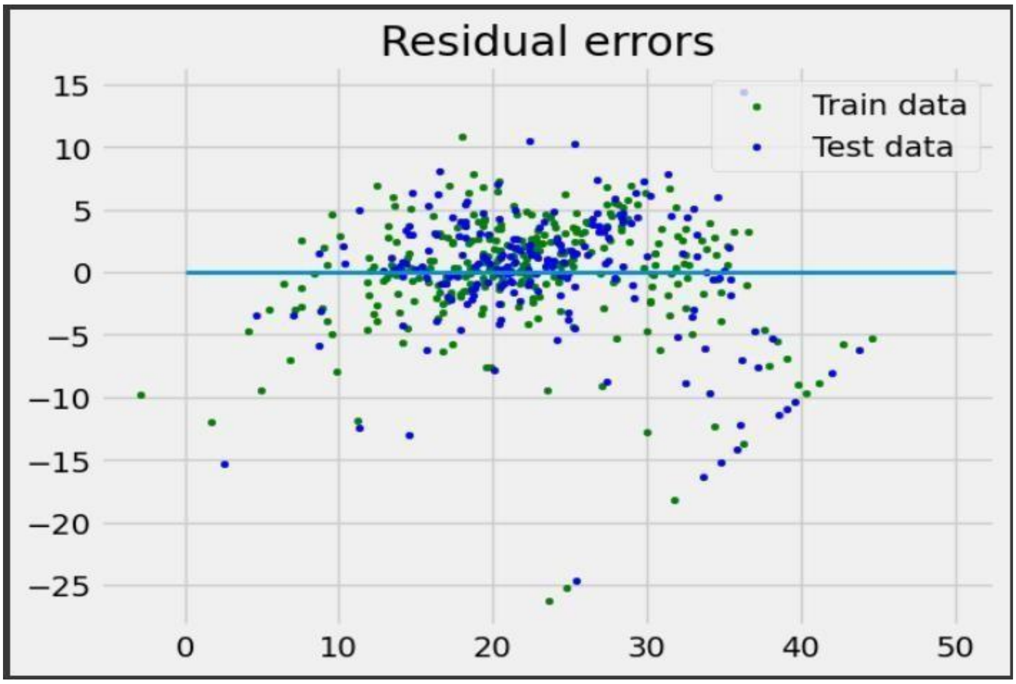
c) Regression Coefficients and Variance Score:

```
# regression coefficients
print('Coefficients: ', reg.coef_)

# variance score: 1 means perfect prediction
print('Variance score: {}'.format(reg.score(X_test, y_test)))
```

```
Coefficients:  [-8.95714048e-02  6.73132853e-02  5.04649248e-02  2.18579583e+00
 -1.72053975e+01  3.63606995e+00  2.05579939e-03 -1.36602886e+00
  2.89576718e-01 -1.22700072e-02 -8.34881849e-01  9.40360790e-03
 -5.04008320e-01]
Variance score: 0.720905667266174
```

d) Plotting:

```
# plot for residual error

# setting plot style
plt.style.use('fivethirtyeight')

# plotting residual errors in training data
plt.scatter(reg.predict(X_train),
            reg.predict(X_train) - y_train,
            color="green", s=10,
            label='Train data')

# plotting residual errors in test data
plt.scatter(reg.predict(X_test),
            reg.predict(X_test) - y_test,
            color="blue", s=10,
            label='Test data')

# plotting line for zero residual error
plt.hlines(y=0, xmin=0, xmax=50, linewidth=2)

# plotting legend
plt.legend(loc='upper right')

# plot title
plt.title("Residual errors")

# method call for showing the plot
plt.show()
```



Practical: 11

Aim: Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

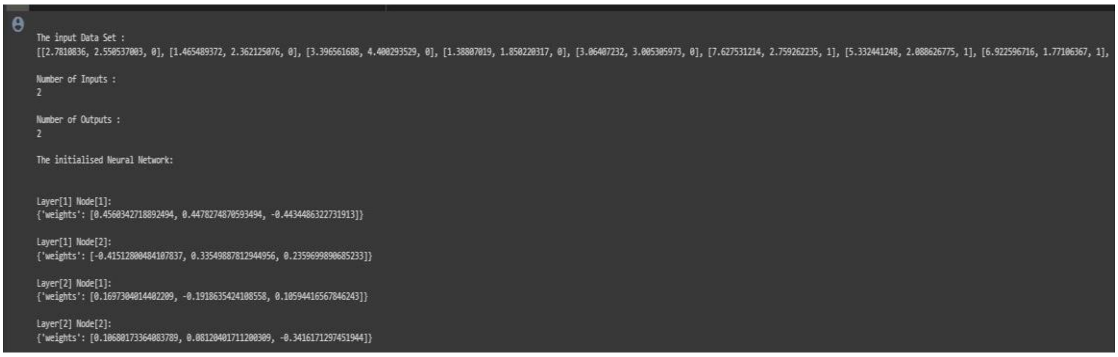
Code:

a) Artificial Neural Network:

```
import random
from math import exp
from random import seed

# Initialize a network

def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights':[random.uniform(-0.5,0.5) for i in range(n_inputs + 1)]} for
i in range(n_hidden)]
    network.append(hidden_layer)
    output_layer = [{'weights':[random.uniform(-0.5,0.5) for i in range(n_hidden + 1)]} for
i in range(n_outputs)]
    network.append(output_layer)
    i= 1
    print("\n The initialised Neural Network:\n")
    for layer in network:
        j=1
        for sub in layer:
            print("\n Layer[%d] Node[%d]:\n" %(i,j),sub)
            j=j+1
        i=i+1
    return network

# Calculate neuron activation (net) for an input

def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation to sigmoid function
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
```

```
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()

        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])

        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])


# Update network weights with error
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1]
        if i != 0:
            inputs = [neuron['output'] for neuron in network[i - 1]]
        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]
            neuron['weights'][-1] += l_rate * neuron['delta']


# Train a network for a fixed number of epochs
def train_network(network, train, l_rate, n_epoch, n_outputs):

    print("\n Network Training Begins:\n")

    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            sum_error += sum([(expected[i]-outputs[i])**2 for i in range(len(expected))])
```

```
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))

    print("\n Network Training Ends:\n")


#Test training backprop algorithm
seed(2)
dataset = [[2.7810836,2.550537003,0],
    [1.465489372,2.362125076,0],
    [3.396561688,4.400293529,0],
    [1.38807019,1.850220317,0],
    [3.06407232,3.005305973,0],
    [7.627531214,2.759262235,1],
    [5.332441248,2.088626775,1],
    [6.922596716,1.77106367,1],
    [8.675418651,-0.242068655,1],
    [7.673756466,3.508563011,1]]

print("\n The input Data Set :\n",dataset)
n_inputs = len(dataset[0]) - 1
print("\n Number of Inputs :\n",n_inputs)
n_outputs = len(set([row[-1] for row in dataset]))
print("\n Number of Outputs :\n",n_outputs)

#Network Initialization
network = initialize_network(n_inputs, 2, n_outputs)

# Training the Network
train_network(network, dataset, 0.5, 20, n_outputs)


print("\n Final Neural Network :")

i= 1
for layer in network:
    j=1
    for sub in layer:
        print("\n Layer[%d] Node[%d]:\n" %(i,j),sub)
        j=j+1
    i=i+1
```



```
Network Training Begins:

>epoch=0, lrate=0.500, error=5.278
>epoch=1, lrate=0.500, error=5.122
>epoch=2, lrate=0.500, error=5.006
>epoch=3, lrate=0.500, error=4.875
>epoch=4, lrate=0.500, error=4.700
>epoch=5, lrate=0.500, error=4.466
>epoch=6, lrate=0.500, error=4.176
>epoch=7, lrate=0.500, error=3.838
>epoch=8, lrate=0.500, error=3.469
>epoch=9, lrate=0.500, error=3.089
>epoch=10, lrate=0.500, error=2.716
>epoch=11, lrate=0.500, error=2.367
>epoch=12, lrate=0.500, error=2.054
>epoch=13, lrate=0.500, error=1.780
>epoch=14, lrate=0.500, error=1.546
>epoch=15, lrate=0.500, error=1.349
>epoch=16, lrate=0.500, error=1.184
>epoch=17, lrate=0.500, error=1.045
>epoch=18, lrate=0.500, error=0.929
>epoch=19, lrate=0.500, error=0.831

Network Training Ends:


Final Neural Network :

Layer[1] Node[1]:
{'weights': [0.8642508164347664, -0.8497601716670761, -0.8668929014392035], 'output': 0.9295587965836384, 'delta': 0.005645382825629247}

Layer[1] Node[2]:
{'weights': [-1.2934302410111027, 1.7109363237151511, 0.7125327507327331], 'output': 0.04760703296164143, 'delta': -0.005928559978815065}

Layer[2] Node[1]:
{'weights': [-1.3098359335096292, 2.16462207144596, -0.3079052288835877], 'output': 0.1989556395205846, 'delta': -0.03170801648036036}

Layer[2] Node[2]:
{'weights': [1.5506793402414165, -2.11315950446121, 0.1333585709422027], 'output': 0.8095042653312078, 'delta': 0.029375796661413225}
```

b) Predict:

```python
from math import exp

# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights)-1):
        activation += weights[i] * inputs[i]
    return activation

# Transfer neuron activation
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))

# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs

# Make a prediction with a network
def predict(network, row):
    outputs = forward_propagate(network, row)
    return outputs.index(max(outputs))

# Test making predictions with the network
dataset = [[2.7810836,2.550537003,0],
    [1.465489372,2.362125076,0],
    [3.396561688,4.400293529,0],
    [1.38807019,1.850220317,0],
    [3.06407232,3.005305973,0],
    [7.627531214,2.759262235,1],
    [5.332441248,2.088626775,1],
    [6.922596716,1.77106367,1],
    [8.675418651,-0.242068655,1],
    [7.673756466,3.508563011,1]]
#network = [[{'weights': [-1.482313569067226, 1.8308790073202204, 1.078381922048799]}, {'weights': [0.23244990332399884, 0.3621998343835864, 0.40289821191094327]}],
#    [{'weights': [2.5001872433501404, 0.7887233511355132, -1.1026649757805829]}, {'weights': [-2.429350576245497, 0.8357651039198697, 1.0699217181280656]}]]
for row in dataset:
    prediction = predict(network, row)
    print('Expected=%d, Got=%d' % (row[-1], prediction))
```

```
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
```