KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Narayanguda, Hyderabad – 500029

DEPARTMENT OF CSE (DS)

LAB MANUAL

ANN & DEEP LEARNING LAB

(21CD503PC)

B.Tech. III YEAR I SEM (RKR21)

ACADEMIC YEAR 2024-25

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTE)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad

Certificate

This is to certify that following is a Bonafide Record of the workbook task done by ____________________________________bearing Roll No_________________ of ________ Branch of ____________ year B.Tech Course in the _______________________________________ Subject during the Academic year _____________& ___________under our supervision.

Number of week tasks completed: ____

Signature of Staff Member Incharge

Signature of Head of the Dept.

Signature of Internal Examiner

Signature of External Examiner

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTE)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad

Daily Laboratory Assessment Sheet

Name of the Lab: Name of the Student:

Class: HT. No:

S.No.	Name of the Experiment	Date	Observation Marks (3M)	Record Marks (4M)	Viva Voice Marks (3M)	Total Marks (10M)	Signature of Faculty

INDEX

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTE)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Narayanguda, Hyderabad – 500029

Department of Computer Science & Engineering (DS)

Vision of the Institution:

To be the fountain head of latest technologies, producing highly skilled, globally competent engineers.

Mission of the Institution:

- To provide a learning environment that inculcates problem solving skills, professional, ethical responsibilities, lifelong learning through multi modal platforms and prepare students to become successful professionals.
- To establish Industry Institute Interaction to make students ready for the industry.
- To provide exposure to students on latest hardware and software tools.
- To promote research based projects/activities in the emerging areas of technology convergence.
- To encourage and enable students to not merely seek jobs from the industry but also to create new enterprises
- To induce a spirit of nationalism which will enable the student to develop, understand India's challenges and to encourage them to develop effective solutions.
- To support the faculty to accelerate their learning curve to deliver excellent service to students

Department of Computer Science & Engineering (DS)

Vision of the Department:

To be among the region's premier teaching and research Computer Science and Engineering departments producing globally competent and socially responsible graduates in the most conducive academic environment.

Mission of the Department:

- To provide faculty with state of the art facilities for continuous professional development and research, both in foundational aspects and of relevance to emerging computing trends.
- To impart skills that transform students to develop technical solutions for societal needs and inculcate entrepreneurial talents.
- To inculcate an ability in students to pursue the advancement of knowledge in various specializations of Computer Science and Engineering and make them industry-ready.
- To engage in collaborative research with academia and industry and generate adequate resources for research activities for seamless transfer of knowledge resulting in sponsored projects and consultancy.
- To cultivate responsibility through sharing of knowledge and innovative computing solutions that benefits the society-at-large.
- To collaborate with academia, industry and community to set high standards in academic excellence and in fulfilling societal responsibilities.

Department of Computer Science & Engineering(DS)

PROGRAM OUTCOMES (POs)

PO1: Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct Investigations of Complex Problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: An ability to analyze the common business functions to design and develop appropriate Computer Science solutions for social upliftment.

PSO2: Shall have expertise on the evolving technologies like Python, Machine Learning, Deep Learning, Internet of Things (IOT), Data Science, Full stack development, Social Networks, Cyber Security, Big Data, Mobile Apps, CRM, ERP etc.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will have successful careers in computer related engineering fields or will be able to successfully pursue advanced higher education degrees.

PEO2: Graduates will try and provide solutions to challenging problems in their profession by applying computer engineering principles.

PEO3: Graduates will engage in life-long learning and professional development by rapidly adapting changing work environment.

PEO4: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTE)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad

B.Tech. in COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

III Year I Semester Syllabus
(RKR21) ANN & DEEP LEARNING
LAB (21CD503PC)

L	T	P	C
0	0	3	1.5

Prerequisites/ Co-requisites:

1. PP207ES - Python Programming Lab Course

Course Objectives: The course will help to

1. Understand Tensor Flow fundamentals
2. Understand the concept of building MLP Models with and without Pred-defined models.
3. Acquire knowledge on building ANN model for image classification.
4. Gain knowledge on Image processing and analysis with CNN.
5. Understand the concepts of building RNN models.

Course Outcomes: The student will be able to

1. Gain proficiency in manipulating tensors, performing mathematical operations, and other fundamental operations on Tensors.
2. Implement MLP Models for tabular data.
3. Develop ANN model for image classification.
4. Develop image classification model using CNN.
5. Implement Sequence learning with RNN, LSTM & GRU models.

List of Experiments:

1. Create Tensors and perform basic operations with tensors.
2. Create Tensors and apply split & merge operations and statistics operations.
3. Design single unit perceptron for classification of iris dataset without using pre-defined models.
4. Design, train and test the MLP for tabular data and verify various activation functions and optimizers tensor flow.
5. Design and implement to classify 32x32 images using MLP using tensorflow/keras and check the accuracy.
6. Design and implement a CNN model to classify multi category JPG images with tensorflow / keras and check accuracy. Predict labels for new images.
7. Design and implement a CNN model to classify multi category tiff images with tensorflow / keras and check the accuracy. Check whether your model is overfit / underfit / perfect fit and apply the techniques to avoid overfit and underfit like regulizers, dropouts etc.
8. Implement a CNN architectures (LeNet, Alexnet, VGG, etc) model to classify

multi category Satellite images with tensorflow / keras and check the accuracy. Check whether your model is overfit / underfit / perfect fit and apply the techniques to avoid overfit and underfit.

9. Design and implement a simple RNN model with tensorflow / keras and check accuracy.
10. Design and implement LSTM model with tensorflow / keras and check accuracy.
11. Design and implement GRU model with tensorflow / keras and check accuracy.

TEXT BOOKS:

1. Beginning Deep Learning with TensorFlow: Work with Keras, MNIST DataSets, and Advanced Neural Networks by Liangqu Long, Xiangming Zeng, A Press, 2022.
2. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition, by Aurélien Géron,O'Reilly Publications,2022.

REFERENCE BOOKS:

1. Artificial Intelligence Fundamentals and Applications- Cherry Bhargava and Pardeep Kumar Sharma, 1st Edition, CRC Press, 2022.
2. Deep Learning Methods and Applications by Li Deng, Dong Yu, Now Publishers Inc, 2014.

Experiment – 1

Aim: Exploratory Data Analysis and Preprocessing on a dataset

Dataset : California Housing Prices Dataset

```
import warnings
warnings.filterwarnings('ignore')
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv('../input/california-housing-prices/housing.csv')
df.head()
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
```

```
df.shape
```

```
(20640, 10)
```

```
df.columns
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
```

```
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Finding Nulls

```
df.isnull().sum()
```

```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

df.isnull().sum()# Handling Nulls

```
df['total_bedrooms'].mean(),df['total_bedrooms'].median(),df['total_bedrooms'
].mode()
```

```
(537.8705525375618,
 435.0,
 0    280.0
 dtype: float64)
```

```
df['total_bedrooms'].fillna(df['total_bedrooms'].median(), inplace=True)
```

Cross checking Nulls

```
df.isnull().sum()
```

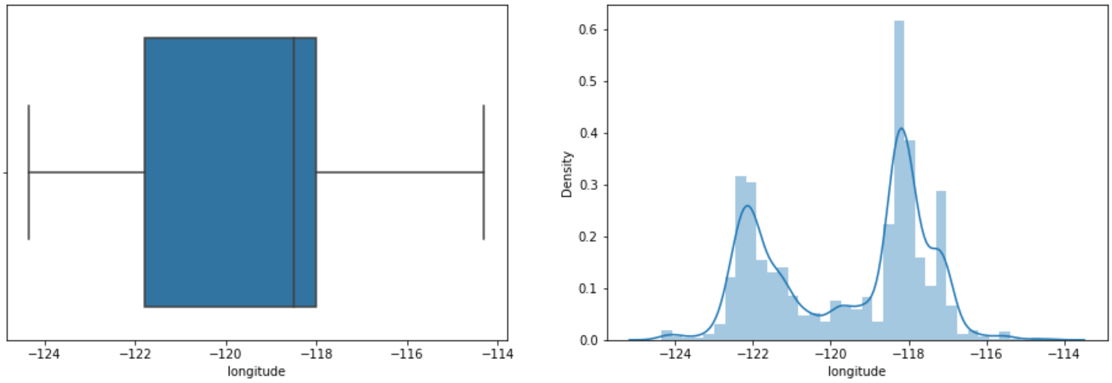
```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

Outliers Detection

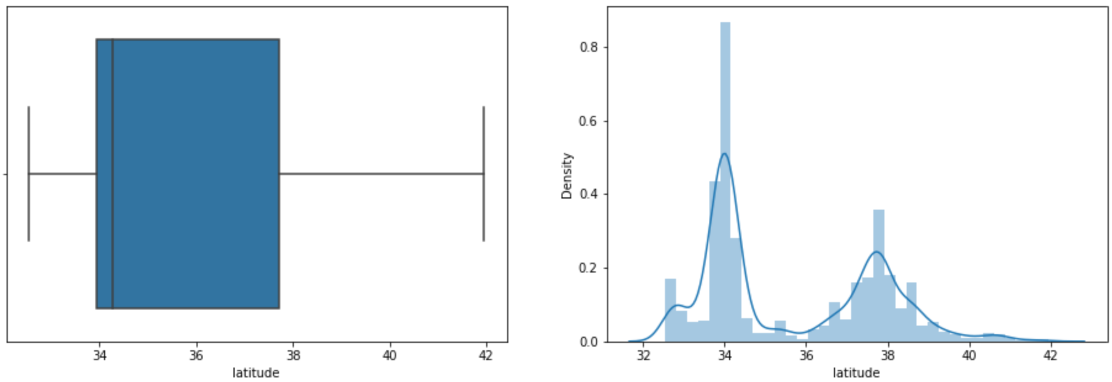
```
#Outlier Detection Using Box plot
plt.figure(figsize=(16, 5))
plt.subplot(1,2,1)
plt1 = sns.boxplot(df['longitude'])
```

x

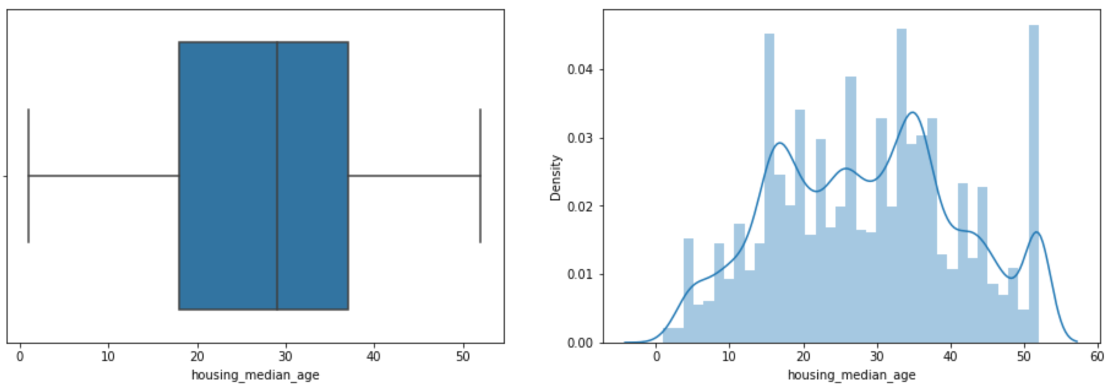
```
plt.subplot(1,2,2)
plt2 = sns.distplot(df['longitude'])
```



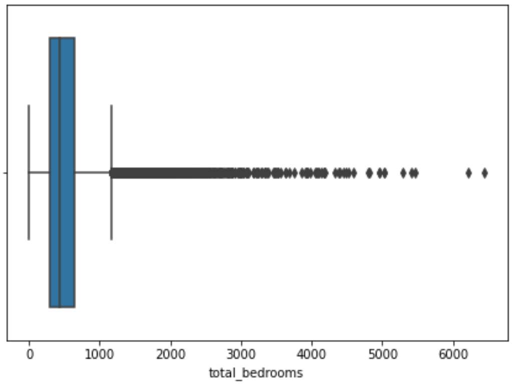

```
#Outlier Detection Using Box plot
plt.figure(figsize=(16, 5))
plt.subplot(1,2,1)
plt1 = sns.boxplot(df['latitude'])
plt.subplot(1,2,2)
plt2 = sns.distplot(df['latitude'])
```

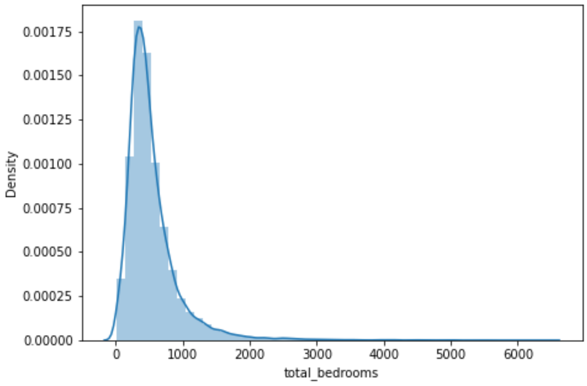



```
#Outlier Detection Using Box plot
plt.figure(figsize=(16, 5))
plt.subplot(1,2,1)
plt1 = sns.boxplot(df['housing_median_age'])
plt.subplot(1,2,2)
plt2 = sns.distplot(df['housing_median_age'])
```

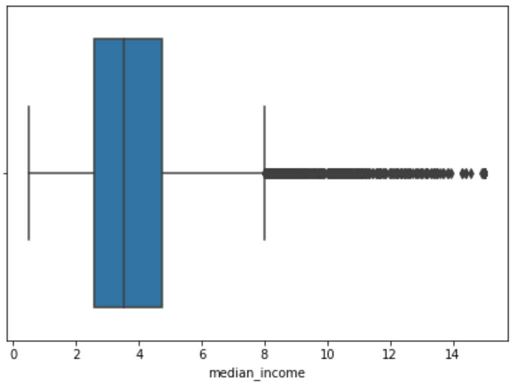


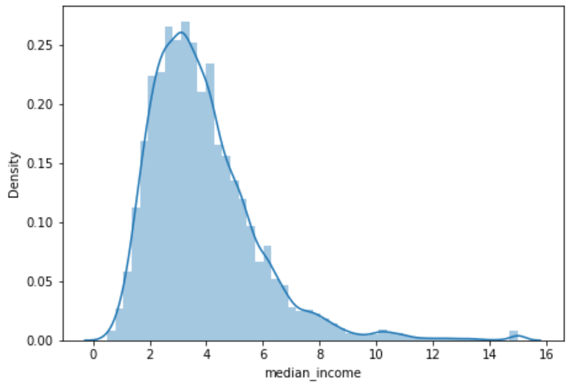
```
#Outlier Detection Using Box plot
plt.figure(figsize=(16, 5))
plt.subplot(1,2,1)
plt1 = sns.boxplot(df['total_bedrooms'])
plt.subplot(1,2,2)
plt2 = sns.distplot(df['total_bedrooms'])
```



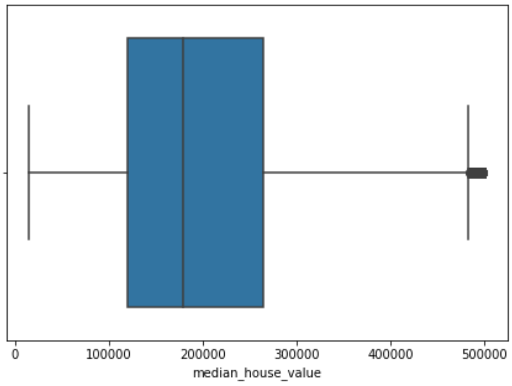


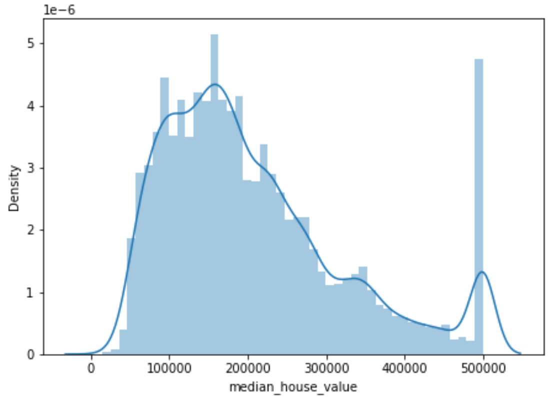

```
#Outlier Detection Using Box plot
plt.figure(figsize=(16, 5))
plt.subplot(1,2,1)
plt1 = sns.boxplot(df['median_income'])
plt.subplot(1,2,2)
plt2 = sns.distplot(df['median_income'])
```






```
#Outlier Detection Using Box plot
plt.figure(figsize=(16, 5))
plt.subplot(1,2,1)
plt1 = sns.boxplot(df['median_house_value'])
plt.subplot(1,2,2)
plt2 = sns.distplot(df['median_house_value'])
```





Finding correlation

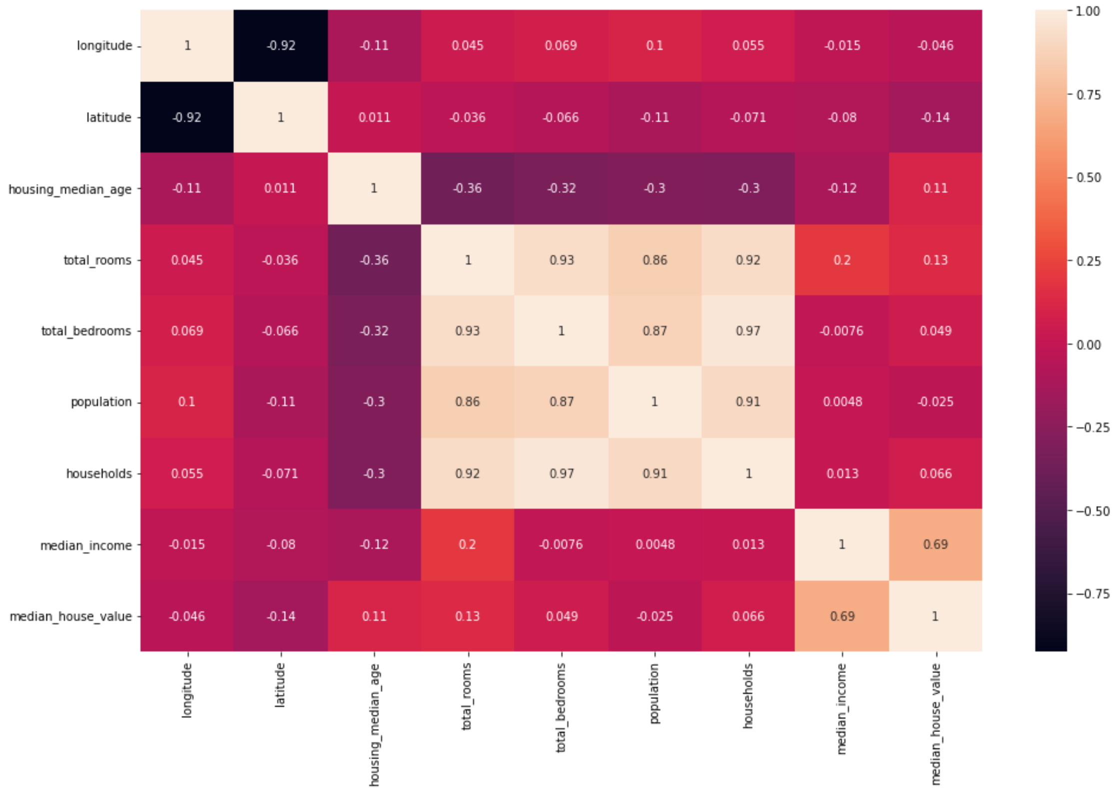
```
df.corr()
```

```
                    longitude  latitude  housing_median_age  total_rooms  \
longitude            1.000000 -0.924664           -0.108197     0.044568
latitude            -0.924664  1.000000            0.011173    -0.036100
housing_median_age  -0.108197  0.011173            1.000000    -0.361262
total_rooms          0.044568 -0.036100           -0.361262     1.000000
total_bedrooms       0.069120 -0.066484           -0.319026     0.927058
population           0.099773 -0.108785           -0.296244     0.857126
households           0.055310 -0.071035           -0.302916     0.918484
median_income       -0.015176 -0.079809           -0.119034     0.198050
median_house_value  -0.045967 -0.144160            0.105623     0.134153

                    total_bedrooms  population  households  median_income  \
longitude                 0.069120    0.099773    0.055310      -0.015176
latitude                 -0.066484   -0.108785   -0.071035      -0.079809
housing_median_age       -0.319026   -0.296244   -0.302916      -0.119034
total_rooms               0.927058    0.857126    0.918484       0.198050
total_bedrooms            1.000000    0.873535    0.974366      -0.007617
population                0.873535    1.000000    0.907222       0.004834
households                0.974366    0.907222    1.000000       0.013033
median_income            -0.007617    0.004834    0.013033       1.000000
median_house_value        0.049457   -0.024650    0.065843       0.688075

                    median_house_value
longitude                    -0.045967
latitude                     -0.144160
housing_median_age            0.105623
total_rooms                   0.134153
total_bedrooms                0.049457
population                   -0.024650
households                    0.065843
median_income                 0.688075
median_house_value            1.000000
```

```
plt.figure(figsize = (16, 10))
sns.heatmap(df.corr(),annot=True)
```



```
<AxesSubplot:>
```




```
dfc=df.corr()
dfc["median_house_value"].sort_values(ascending=False)
```

```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049457
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

```
#data.median_income.values.reshape(-1,1)
x=df.median_income.values
print(x)
x=df.median_income.values.reshape(-1,1)
x
```

```
[8.3252 8.3014 7.2574 ... 1.7    1.8672 2.3886]
```

```
array([[8.3252],
       [8.3014],
       [7.2574],
       ...,
       [1.7   ],
       [1.8672],
       [2.3886]])
```

xiv

```
#data.median_house_value.reshape(-1,1)
y=df.median_house_value.values
print(y)
```

```
y=df.median_house_value.values.reshape(-1,1)
y
```

```
[452600. 358500. 352100. ...  92300.  84700.  89400.]

array([[452600.],
       [358500.],
       [352100.],
       ...,
       [ 92300.],
       [ 84700.],
       [ 89400.]])
```

Splitting train and test datasets

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.2,shuffle=True)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((16512, 1), (4128, 1), (16512, 1), (4128, 1))
```

Building model: Simple Linear Regression

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(x_train,y_train)
```

```
LinearRegression()
```

Evaluate the model (intercept and slope)

```
lm.coef_,lm.intercept_

y_pred= lm.intercept_+lm.coef_*x_train
y_pred
```

```
array([[147926.02519282],
       [119024.2326875 ],
       [142678.73770954],
       ...,
       [123558.25613292],
       [187985.60308589],
       [331585.25824256]])
```

Predicting the test set result

```
#Prediction for test dataset
y_pred=lm.predict(x_test)
y_pred
```

```
array([[253776.91109374],
       [144851.89890094],
       [118498.66971223],
       ...,
       [145327.40825953],
```

```
       [151012.66488887],
       [ 84399.64334032]])
```

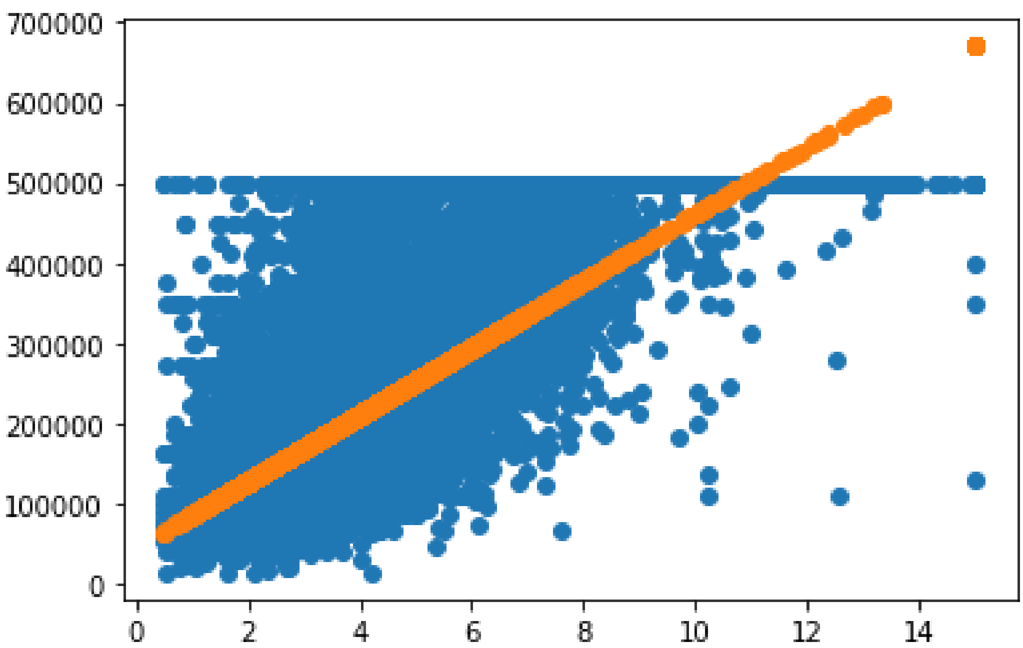
Performance Measures

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
mean_absolute_error(y_test,y_pred),np.sqrt(mean_squared_error(y_test,y_pred))
```

```
(62523.90980063932, 83533.96149114512)
```

```
plt.scatter(x,y)
plt.scatter(x_test,y_pred)
```

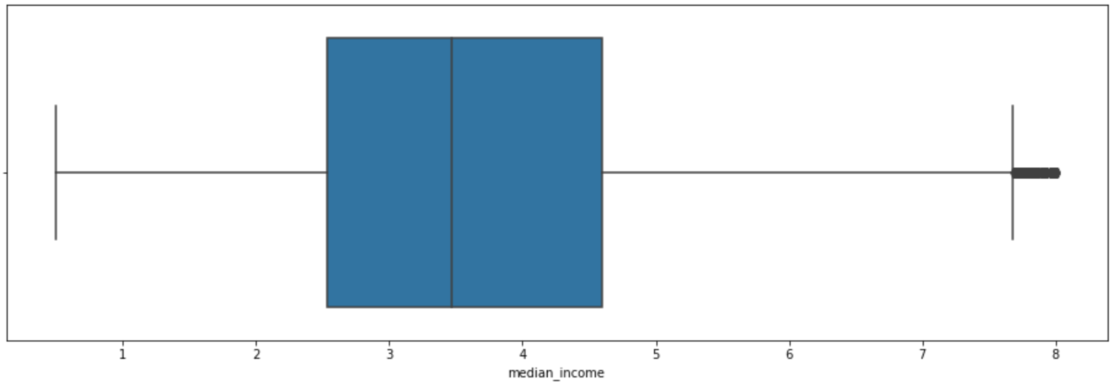
```
<matplotlib.collections.PathCollection at 0x7bed78f79e50>
```



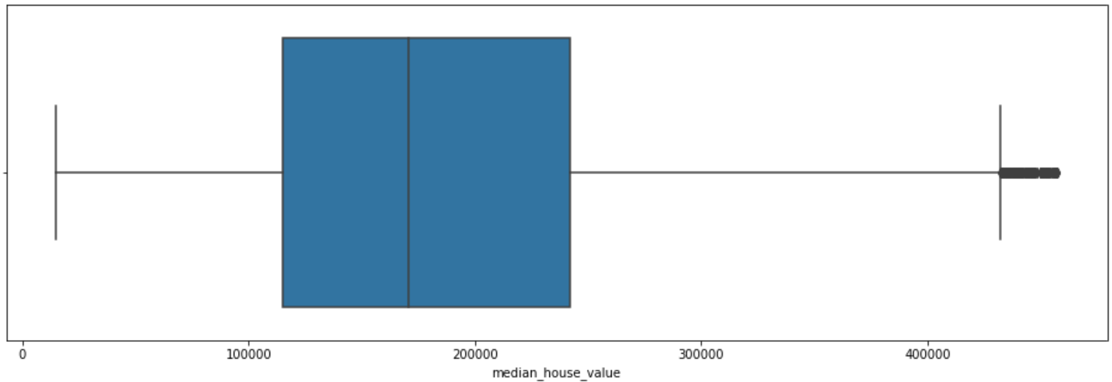
Outlier Treatment

```
def outliersTreatment(df1, attr):
    percentile25 = df1[attr].quantile(0.25)
    percentile75 = df1[attr].quantile(0.75)
    iqr = percentile75 - percentile25
    print(percentile25)
    print(percentile75)
    print(iqr)
    upper_limit = percentile75 + 1.5 * iqr
    lower_limit = percentile25 - 1.5 * iqr
    df1[df1[attr] > upper_limit]
    df1[df1[attr] < lower_limit]
    df1 = df1[(df1[attr] <= upper_limit) & (df1[attr] >= lower_limit)]
    plt.figure(figsize=(16, 5))
    sns.boxplot(df1[attr])
    plt.show()
    global df
    df = df1

outliersTreatment(df, 'median_income')
outliersTreatment(df, 'median_house_value')
```

```
2.5633999999999997
4.74325
2.17985
```




```
117600.0
253450.0
135850.0
```




```
#data.median_income.values.reshape(-1,1)
x=df.median_income.values
print(x)
x=df.median_income.values.reshape(-1,1)
x
```

```
[7.2574 5.6431 3.8462 ... 1.7    1.8672 2.3886]

array([[7.2574],
       [5.6431],
       [3.8462],
       ...,
       [1.7   ],
       [1.8672],
       [2.3886]])
```

```
#data.median_house_value.reshape(-1,1)
y=df.median_house_value.values
print(y)
y=df.median_house_value.values.reshape(-1,1)
y
```



```
[352100. 341300. 342200. ...  92300.  84700.  89400.]
```

```
array([[352100.],
       [341300.],
       [342200.],
       ...,
       [ 92300.],
       [ 84700.],
       [ 89400.]])
```

```
x.shape,y.shape
```

```
((19237, 1), (19237, 1))
```

Splitting train and test datasets

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = 
train_test_split(x,y,test_size=0.2,shuffle=True)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((15389, 1), (3848, 1), (15389, 1), (3848, 1))
```

Building model: Simple Linear Regression

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(x_train,y_train)
```

```
LinearRegression()
```

Evaluate the model (intercept and slope).

```
lm.coef_,lm.intercept_

y_pred= lm.intercept_+lm.coef_*x_train
y_pred
```

```
array([[247285.6572024 ],
       [196534.28494803],
       [147202.39401263],
       ...,
       [228399.87112215],
       [216568.23874334],
       [233940.17355544]])
```

Predicting the test set result

```
#Prediction for test dataset
y_pred=lm.predict(x_test)
y_pred
```

```
array([[142995.06711156],
       [272757.67931937],
       [189212.74972392],
       ...,
       [263049.37079717],
```

```
        [161188.80689779],
        [130046.72366741]])
```

```
y_test,y_pred
```

```
(array([[102900.],
        [186100.],
        [108500.],
        ...,
        [245800.],
        [151800.],
        [100000.]]),
 array([[142995.06711156],
        [272757.67931937],
        [189212.74972392],
        ...,
        [263049.37079717],
        [161188.80689779],
        [130046.72366741]]))
```

Performance Measures

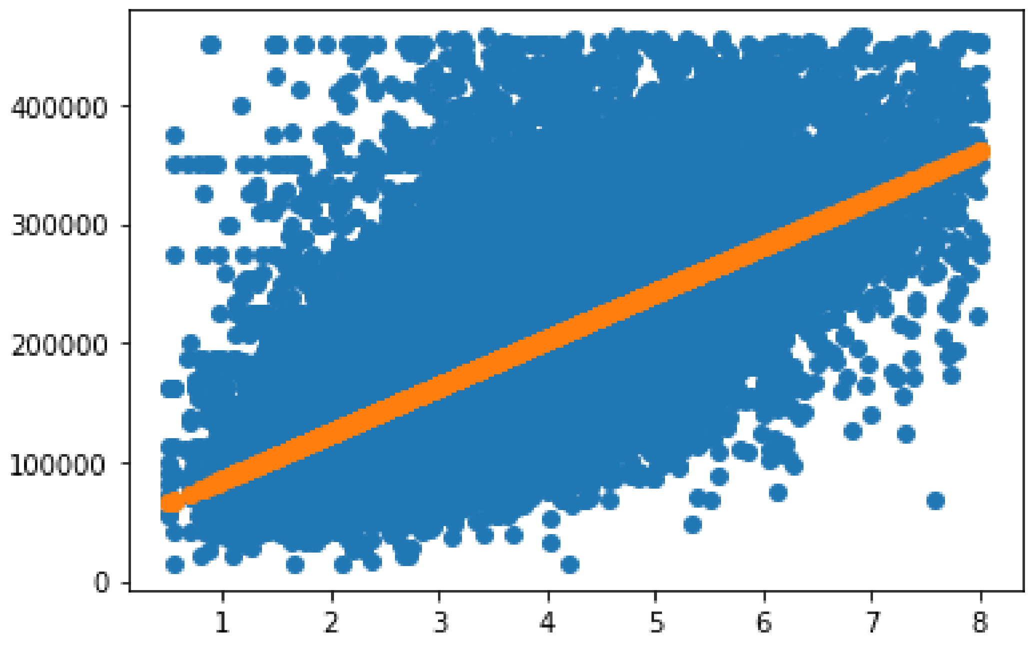
```
from sklearn.metrics import mean_squared_error, mean_absolute_error
mean_absolute_error(y_test,y_pred),np.sqrt(mean_squared_error(y_test,y_pred))
```

```
(53387.26749124496, 69087.22438535451)
```

Visualize the training set and testing set using Matplotlib, Seaborn.

```
plt.scatter(x,y)
plt.scatter(x_test,y_pred)
```

```
<matplotlib.collections.PathCollection at 0x7bed78f19550>
```



Polynomial Regression

```
from sklearn.preprocessing import PolynomialFeatures
trans = PolynomialFeatures(degree=4)
x = trans.fit_transform(x)
x.shape
```

```
(19237, 5)
```

```
from sklearn.model_selection import train_test_split
x_train,x_test, y_train,y_test = train_test_split(x,y,test_size=0.2)
x_train.shape,y_train.shape, x_test.shape,y_test.shape
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(x_train,y_train)
```

```
LinearRegression()
```

```
y.min(),y.max()
```

```
(14999.0, 457200.0)
```

```
# Prediction for test dataset
y_test_pred=lm.predict(x_test)
print('MAE of Polynomial regressions is ..
',mean_absolute_error(y_test,y_test_pred))
```

```
MAE of Polynomial regressions is ..  53803.75640734392
```

```
print(x_train[:,1].shape)
```

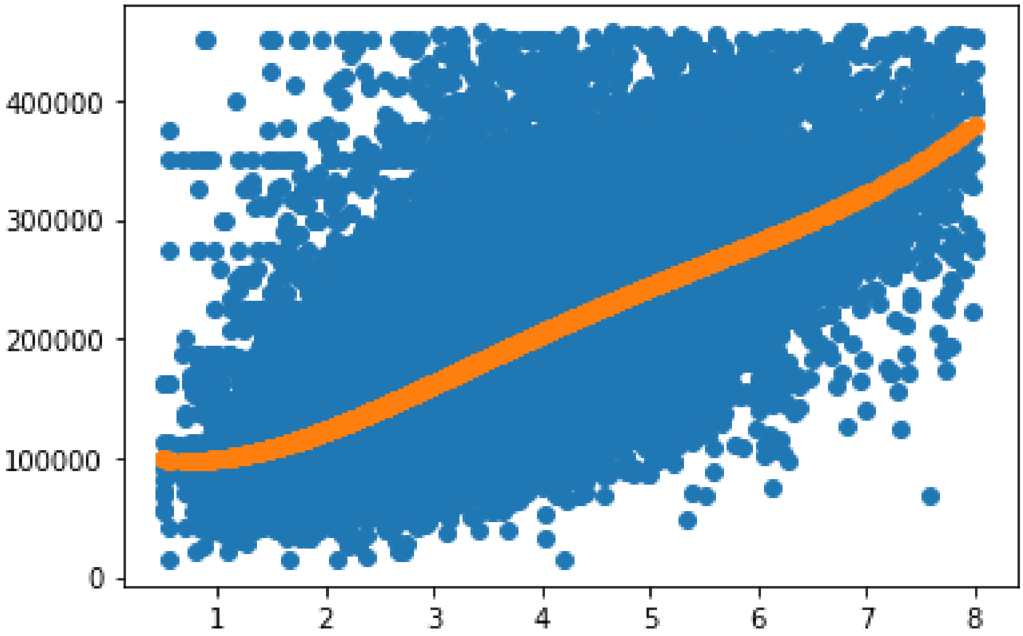
```
(15389,)
```

```
print(y_test_pred.flatten().shape)
```

```
(3848,)
```

```
plt.scatter(x[:,1],y)
plt.scatter(x_test[:,1].flatten(),y_test_pred.flatten())
```

```
<matplotlib.collections.PathCollection at 0x7bed78bcee50>
```



Multiple Regression

```
#Select features for multiple regression
x=df[['median_income','total_rooms','housing_median_age','latitude']]
y=df.median_house_value.values.reshape(-1,1)

x.max(),x.min(),y.max(),y.min()
```

```
(median_income              8.0113
 total_rooms            39320.0000
 housing_median_age        52.0000
 latitude                  41.9500
 dtype: float64,
 median_income          0.4999
 total_rooms            2.0000
 housing_median_age     1.0000
 latitude              32.5400
 dtype: float64,
 457200.0,
 14999.0)
```

```
from sklearn.model_selection import train_test_split
x_train,x_test, y_train,y_test = train_test_split(x,y,test_size=0.2)
x_train.shape,y_train.shape, x_test.shape,y_test.shape
```

```
((15389, 4), (15389, 1), (3848, 4), (3848, 1))
```

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(x_train,y_train)
```

```
LinearRegression()
```

```
#Prediction for test dataset
y_pred=lm.predict(x_test)
y_pred
```

```
array([[ 79502.39366596],
       [251664.74335574],
       [ 98843.47492845],
       ...,
       [139778.50024834],
       [ 97620.36348617],
       [184663.83181207]])
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
mean_absolute_error(y_test,y_pred),np.sqrt(mean_squared_error(y_test,y_pred))
```

```
(51655.75375902168, 68048.3299317812)
```

Platform Used: Kaggle

Conclusion: EDA on California Housing Prices Dataset is done

Experiment - 2

Aim: Create Tensors and perform basic operations with tensors.

Problem Statement:
To create tensors of any dimension and to apply basic operations to those tensors.

Dataset:
We create our own set of tensors using numpy arrays. Hence, there is no need to use any pre-built dataset.

```
import tensorflow as tf
import numpy as np
# 1. Creating tensors using python lists & numpy arrays
tlist = np.array([[10, 20, 30], [40, 50, 60]])
tnumpy = np.array([[79, 89, 99], [109, 119, 129]])
# Converting lists and numpy arrays to TensorFlow tensors
tensor = tf.convert_to_tensor(tnumpy, dtype=tf.int32)
# 2. Creating tensors filled with zeros and ones using TensorFlow
zerotensor = tf.zeros((2, 3), dtype=tf.int32)
onestensor= tf.ones((2, 3), dtype=tf.int32)
# 3. Creating two tensors and performing basic arithmetic operations
a = tf.convert_to_tensor(np.array([[10, 20, 30], [40, 50, 60]]), dtype=tf.int32)
b = tf.convert_to_tensor(np.array([[79, 89, 99], [109, 119, 129]]), dtype=tf.int32)
tensor_add_tf = tf.add(a, b)
tensor_sub_tf = tf.subtract(a, b)
tensor_mul_tf = tf.multiply(a, b)
tensor_div_tf = tf.divide(a, b)
# Display the results
print("Tensor from numpy array:\n", tnumpy)
print("Zero tensor:\n", zerotensor)
print("Ones tensor:\n", onestensor)
print("Addition:\n", tensor_add_tf)
print("Subtraction:\n", tensor_sub_tf)
print("Multiplication:\n", tensor_mul_tf)
print("Division:\n", tensor_div_tf)
```

Output:

```
Tensor from numpy array:
 [[ 79  89  99]
 [109 119 129]]
Zero tensor:
 tf.Tensor(
[[0 0 0]
 [0 0 0]], shape=(2, 3), dtype=int32)
Ones tensor:
 tf.Tensor(
```

```
[[1 1 1]
 [1 1 1]], shape=(2, 3), dtype=int32)
Addition:
 tf.Tensor(
[[ 89 109 129]
 [149 169 189]], shape=(2, 3), dtype=int32)
Subtraction:
 tf.Tensor(
[[-69 -69 -69]
 [-69 -69 -69]], shape=(2, 3), dtype=int32)
Multiplication:
 tf.Tensor(
[[ 790 1780 2970]
 [4360 5950 7740]], shape=(2, 3), dtype=int32)
Division:
 tf.Tensor(
[[0.12658228 0.2247191  0.3030303 ]
 [0.36697248 0.42016807 0.46511628]], shape=(2, 3), dtype=float64)
```

Platform used: Kaggle

Conclusion:

Created different types of tensors and performd basic mathematical operations.

Experiment - 3

Aim: Create Tensors and apply split & merge operations and statistics operations.

Problem Statement:
To create tensors of any dimension and to apply split , merge and statistical operations to those tensors.

Dataset:
We create our own set of tensors using numpy arrays. Hence, there is no need to use any pre-built dataset

```
#import tensorflow as tf

a=tf.random.normal([4,3,3])
print(a)
```

Output:

```
tf.Tensor(
[[[ 0.8905804   0.11471634 -0.4948614 ]
  [ 1.3300606   0.08043179  0.47385365]
  [-0.01988183  0.0469541   1.1030353 ]]

 [[ 1.254597   -0.1476164   0.4048071 ]
  [-0.15055096 -0.3522078  -0.8308792 ]
  [-0.5762596  -1.7004256  -0.15538406]]

 [[ 0.18634023 -0.193523    0.1674539 ]
  [ 0.60915816  0.85616803 -0.01242118]
  [-1.8685336  -0.567578   -1.8146135 ]]

 [[ 1.3720973  -0.5825495   0.48135617]
  [ 1.527614    1.7224327  -0.8411025 ]
  [ 0.5043328   1.5071727   1.0533932 ]]], shape=(4, 3, 3), dtype=float32)
```

```
a=tf.random.normal([4,35,8])
print(a)
```

Output:

```
tf.Tensor(
[[[ 1.0984889  -0.0946807   0.58538526 ...  1.0094188   0.633665
    0.135887  ]
  [-0.96918595  0.39862677  1.9723798  ...  0.30364797  0.05226479
    0.88370496]
  [ 0.36264858 -0.13501921  0.27693936 ... -0.26362896  0.98305863
   -0.7645232 ]
```

```
...
[ 0.11513168 -0.62268597 -0.05530274 ...  0.5727292  -0.18652977
  1.0058318 ]
[-1.8922123   0.61025983 -1.2823237  ...  1.2926298   0.89289105
 -1.2886316 ]
[ 1.8497883  -0.1807306   2.1537328  ...  1.8228115   0.2646232
  2.5489223 ]]

[[-0.8047426   0.11335723 -0.14152905 ... -0.5521221  -0.23727784
  1.0579133 ]
[-0.7806445   0.7881308   0.5658614  ... -2.4091005  -0.8247843
  0.3393152 ]
[-0.8299016   0.8728088  -0.5984835  ...  0.6614035  -1.0256618
 -0.2806373 ]
...
[ 0.34772408  1.0936401   0.988164   ...  2.170961    0.55737686
  0.5540246 ]
[ 0.20095216 -0.20280902 -1.1819931  ... -1.1123956  -0.27960575
  2.1057963 ]
[-0.28411722 -0.0505016   0.2528023  ... -0.92547953 -1.448679
 -1.0324591 ]]

[[-1.7706002  -0.44989493  0.714208   ... -1.052579   -0.26691645
  1.0000407 ]
[-0.2578996  -1.8586687   1.3850516  ...  0.6570459  -0.8631878
 -1.1230085 ]
[-1.1717957   1.1484773  -1.3279521  ... -0.21805455 -1.13351
 -1.1099452 ]
...
[ 0.49565992  0.141708    1.1941801  ... -1.0459158   0.3001566
 -3.2374089 ]
[-1.1189854  -1.0969083   0.6076548  ... -1.0450063  -0.660058
  0.10754981]
[ 0.29436466  1.352847    0.53958637 ... -0.2840581   0.5769609
  0.6900443 ]]

[[ 0.77308816  0.40611103 -1.5777187  ...  0.23795432  1.080221
 -0.25806275]
[-1.1831563  -0.63176686 -1.27217    ... -0.05211914  0.34412125
 -0.918503  ]
[-0.06475189 -0.09764063 -1.2114547  ... -1.6345634  -0.54436785
  0.8997945 ]
...
[-0.53733337 -0.8896664   0.17078346 ...  1.384932   -0.84078956
 -1.1247495 ]
[-0.5047605  -1.2720212  -1.6711704  ...  1.1172732   0.76035684
 -0.5821203 ]
[ 0.8346725  -2.3445506  -1.4626505  ... -1.4955056   0.26448
  0.5684417 ]]], shape=(4, 35, 8), dtype=float32)
```

```
a=tf.random.normal([4,35,8])
b=tf.random.normal([6,35,8])
tf.concat([a,b],axis=0)
```

Output:

```
<tf.Tensor: shape=(10, 35, 8), dtype=float32, numpy=
array([[[ 1.2555432 , -0.31670693, -1.0082678 , ...,  1.9611647 ,
          0.3943139 , -2.095727  ],
        [ 0.40469077,  0.4894915 , -1.6231725 , ..., -0.9641147 ,
          0.22587025, -1.1948998 ],
        [ 0.16469178, -1.033445  , -0.2833022 , ..., -0.01100344,
         -0.50187445, -0.6275308 ],
        ...,
        [ 0.07266472,  0.84395957,  0.6368043 , ..., -1.1079284 ,
          1.4568437 , -0.37582546],
        [-0.99123234, -0.7154441 ,  0.28629294, ...,  0.48389187,
         -0.59579915, -1.706934  ],
        [-0.5642254 ,  1.0589857 , -0.9566821 , ...,  1.1088558 ,
         -1.4998193 , -0.78981525]],

       [[-0.69488823, -0.5144331 , -0.01512962, ...,  0.24453393,
         -1.9662553 , -0.67358965],
        [ 0.9643352 , -0.21193884,  0.72141004, ...,  0.0877675 ,
         -0.1764446 ,  1.1515707 ],
        [ 0.12142342,  1.0560974 ,  0.47533515, ..., -0.29283518,
          0.31340945,  0.41502517],
        ...,
        [-0.4085273 , -2.243514  , -0.49620977, ...,  0.44525716,
         -0.4351831 ,  1.0949378 ],
        [-0.749936  ,  1.2746722 , -1.0618553 , ..., -0.7259868 ,
          0.26423287,  0.9239967 ],
        [-0.07849403, -0.95287853, -0.4308322 , ..., -0.261055  ,
         -1.0518326 , -0.41503695]],

       [[-1.2203016 ,  1.1935928 , -0.6389236 , ..., -0.9882171 ,
         -0.5784913 , -0.45731625],
        [-0.0883997 , -1.7036376 ,  1.4925758 , ...,  0.37709272,
         -2.0592933 ,  0.17649348],
        [-0.93125707, -0.33716512,  0.96875906, ..., -0.85854083,
         -0.20047358, -1.6733274 ],
        ...,
        [-1.0474776 , -1.1301794 ,  0.05970852, ..., -0.34459046,
          0.6704791 ,  0.4137935 ],
        [ 1.0673779 ,  0.5363273 ,  0.59294   , ...,  0.40347117,
          0.53436214, -0.9268911 ],
        [-1.8290071 ,  0.24969505, -0.522733  , ..., -2.1405778 ,
         -1.0729074 , -0.49823228]],

       ...,
```

```
       [[ 0.6534019 , -0.18964013,  0.3762199 , ..., -1.6209399 ,
          1.1747507 ,  0.11061062],
        [-0.24856468, -0.6651757 ,  0.58603   , ...,  0.16929045,
         -1.3459688 ,  0.5337743 ],
        [ 2.6153555 ,  1.1303573 ,  0.4637695 , ...,  0.32921657,
         -1.4879099 , -0.38492602],
        ...,
        [-0.9942837 ,  0.8085243 , -0.5159393 , ..., -0.34137818,
         -0.18782227,  0.50741357],
        [ 1.8813016 , -1.0230325 ,  1.0370253 , ..., -0.88545126,
         -1.6625472 ,  0.6168487 ],
        [ 0.20705721,  0.4478537 ,  0.46624738, ...,  2.5363991 ,
         -0.38055584, -1.4044727 ]],

       [[ 1.6356124 , -0.28398493, -1.3682303 , ...,  0.2719561 ,
          0.7500563 , -0.8875366 ],
        [-0.86430126,  0.9333256 , -0.72825813, ...,  0.2548329 ,
          1.0956444 , -1.9827089 ],
        [ 0.9785393 ,  0.665242  ,  0.19999638, ...,  2.0284572 ,
          0.52267665,  0.57238734],
        ...,
        [ 1.1416398 ,  0.13827191, -1.2880509 , ...,  0.6920185 ,
          1.0375875 ,  0.5261642 ],
        [-1.8825974 ,  0.4086166 , -1.1884203 , ..., -2.0752661 ,
          0.7622692 , -0.4378909 ],
        [ 1.0549271 ,  0.7781699 , -0.30558518, ...,  0.6748719 ,
         -0.05009146,  0.85268015]],

       [[-3.0163553 , -0.6225862 ,  0.89649385, ...,  0.5210352 ,
          0.69692296, -1.6794312 ],
        [-1.0766817 ,  0.76763314,  0.59563464, ...,  1.3585321 ,
         -1.0895659 ,  0.62504405],
        [ 0.74883944, -0.8717583 , -0.65139997, ...,  0.969376  ,
         -0.81280935, -2.1115468 ],
        ...,
        [-0.8603186 , -0.36888003, -0.08816122, ...,  1.7989521 ,
         -0.7997413 , -0.8880248 ],
        [-0.21186765,  0.01915451, -1.2938627 , ..., -0.6748072 ,
         -0.53573984, -0.6141491 ],
        [-1.3200476 , -0.9171071 , -2.1730726 , ..., -1.3448918 ,
         -0.0398301 ,  0.05039204]]], dtype=float32)>
```

```
a=tf.random.normal([4,3,3])
b=tf.random.normal([4,3,3])
tf.concat([a,b],axis=2)
```


Output:

```
<tf.Tensor: shape=(4, 3, 6), dtype=float32, numpy=
array([[[ 0.23168969, -0.43365777,  0.14218141, -1.1179597 ,
```

```
        -0.53772277,  0.70866424],
       [ 0.10632905, -0.14440411,  1.7717392 ,  1.2319126 ,
        -0.05477821,  0.76581055],
       [ 0.79354465, -0.167244  ,  0.2806159 , -0.83516526,
         0.6241413 , -0.9913393 ]],

      [[-0.03125782, -0.21546684, -0.8938286 ,  0.19168243,
        -0.9590286 , -0.85315573],
       [-0.42697498,  0.09738661,  1.1696652 , -1.7646469 ,
        -0.5843887 , -0.31920758],
       [-1.336224  , -1.1552929 ,  1.5981982 , -0.35868728,
         0.03856316, -0.24130213]],

      [[-1.3714752 , -0.5499934 ,  0.90046793, -1.0186541 ,
        -0.41757154, -1.2622834 ],
       [-1.0515286 ,  1.4257001 , -0.7057619 , -0.38418287,
        -1.1117151 ,  0.7098658 ],
       [ 0.4344435 ,  1.014777  , -0.3691092 , -0.31997752,
        -1.0659325 , -0.30986696]],

      [[ 0.6872205 ,  0.84968865,  0.31526884,  0.85081035,
        -0.32066497, -0.37094483],
       [-0.89040107, -0.5143081 ,  0.53963023,  0.5973868 ,
        -0.4321172 ,  0.11585514],
       [-0.06855755,  1.3119028 , -0.57899594,  0.7674852 ,
         0.09014878, -0.9920577 ]]], dtype=float32)>
```

```
a=tf.random.normal([35,8])
b=tf.random.normal([35,8])
tf.stack([a,b],axis=2)
```

Output:

```
<tf.Tensor: shape=(35, 8, 2), dtype=float32, numpy=
array([[[ 2.17151642e+00,  4.29239273e-01],
       [-2.35406011e-01,  1.00386953e+00],
       [-9.12864387e-01, -2.79747891e+00],
       [-6.63628995e-01, -5.35706639e-01],
       [ 1.05530119e+00, -1.83044434e-01],
       [ 1.83501351e+00, -4.61971194e-01],
       [ 6.55516148e-01,  6.70880685e-03],
       [-5.93708694e-01, -1.52739263e+00]],

      [[-1.32878780e+00,  1.58456016e+00],
       [-1.33263135e+00,  2.46793181e-01],
       [-8.75160694e-01, -6.83997989e-01],
       [-7.16515124e-01, -3.57058704e-01],
       [ 1.79529238e+00, -9.78378952e-01],
       [ 1.85086459e-01,  1.50557506e+00],
       [ 1.84949785e-01, -9.65789795e-01],
       [-1.14518905e+00, -2.48904184e-01]],
```

```
[[-1.06087744e-01, -6.38656139e-01],
 [-5.25497019e-01,  6.34478629e-01],
 [ 9.39704478e-02, -3.15284282e-01],
 [ 4.14255321e-01,  1.19365811e+00],
 [-1.47832429e+00, -1.23470671e-01],
 [ 5.70028484e-01,  6.07306957e-02],
 [-3.93116802e-01, -2.31790513e-01],
 [-1.78384292e+00, -1.99501097e+00]],

[[-1.21519625e+00,  1.82715452e+00],
 [-6.49635673e-01,  8.27363789e-01],
 [-1.29806137e+00, -9.36737835e-01],
 [-5.28004169e-01,  1.48346794e+00],
 [-9.80769277e-01, -1.52586460e+00],
 [ 7.24541485e-01, -2.50967871e-02],
 [ 1.30125892e+00,  6.66647404e-02],
 [-3.99279594e-01, -1.30095470e+00]],

[[-9.32339653e-02,  1.12108886e+00],
 [ 1.52265906e+00, -8.88176084e-01],
 [ 1.16271091e+00,  5.38560629e-01],
 [ 7.59633303e-01,  5.21555603e-01],
 [-5.77784240e-01,  3.15223575e-01],
 [ 8.74771118e-01, -1.17755282e+00],
 [-2.50738293e-01, -3.68727627e-03],
 [-2.04045248e+00,  7.38162041e-01]],

[[-6.35827243e-01,  6.52419627e-01],
 [ 1.38822988e-01,  6.24868989e-01],
 [ 3.00374806e-01,  1.49343038e+00],
 [-3.56908798e-01,  1.19741619e+00],
 [-2.13064238e-01, -3.97832304e-01],
 [ 1.88916981e+00,  1.28007507e+00],
 [ 3.59102368e-01, -3.03735137e-01],
 [-8.55196953e-01, -1.17482483e+00]],

[[-2.77730465e-01, -2.27541000e-01],
 [-2.44133139e+00,  2.70693719e-01],
 [-1.03194726e+00, -5.65748572e-01],
 [-4.79330756e-02, -1.24450660e+00],
 [-5.00860631e-01,  5.38235068e-01],
 [ 7.89447904e-01,  1.14781439e+00],
 [-2.48752260e+00,  5.36714375e-01],
 [-4.66566622e-01, -1.42536891e+00]],

[[-1.95133045e-01, -3.15707065e-02],
 [ 6.57168388e-01, -2.43496999e-01],
 [-2.64054865e-01, -4.59223270e-01],
 [-2.11492276e+00,  9.80450869e-01],
```

xxx

```
 [ 4.12952185e-01,  1.21827722e+00],
 [ 3.43030721e-01, -6.05416059e-01],
 [ 7.46167243e-01, -4.39031571e-01],
 [ 6.18139505e-01, -4.78007011e-02]],

[[ 1.32481062e+00, -1.06514826e-01],
 [-1.06985962e+00,  2.61199355e-01],
 [ 9.58096623e-01, -3.88856888e-01],
 [-3.05634886e-01, -1.54191300e-01],
 [ 2.46197179e-01, -3.66119832e-01],
 [ 3.02656919e-01,  3.00844222e-01],
 [-1.01455700e+00,  9.62666094e-01],
 [ 1.00707054e-01,  4.98817265e-01]],

[[-7.67557025e-01,  7.51792967e-01],
 [ 1.89353645e-01,  1.17843175e+00],
 [ 1.14325237e+00, -2.03673887e+00],
 [-8.04508328e-01, -1.29078805e+00],
 [-2.99913675e-01,  8.57654393e-01],
 [-1.43937480e+00, -6.36936486e-01],
 [-7.46092081e-01, -7.85110950e-01],
 [ 5.36906421e-01, -1.25366151e+00]],

[[ 9.37284410e-01, -1.07599819e+00],
 [ 4.50383157e-01, -2.20071658e-01],
 [-8.52738619e-02, -2.59448767e-01],
 [-2.37049669e-01, -6.88724458e-01],
 [ 1.12783706e+00,  1.34203684e+00],
 [ 2.05801415e+00, -7.64297605e-01],
 [ 6.97421670e-01, -4.31094289e-01],
 [ 9.12069499e-01,  4.03095409e-02]],

[[-4.74891752e-01,  1.68569386e+00],
 [-7.01138914e-01, -5.83528459e-01],
 [-1.14394236e+00,  6.34161532e-01],
 [ 7.71305203e-01,  1.78401113e-01],
 [ 9.47572470e-01,  1.75503683e+00],
 [-1.17081738e+00, -9.78661418e-01],
 [-4.97296095e-01,  4.37276363e-01],
 [ 5.57105616e-02, -1.36493087e+00]],

[[-8.77272487e-01,  6.44834936e-01],
 [ 4.73899692e-01,  7.53314853e-01],
 [ 6.59752071e-01,  1.65540540e+00],
 [-3.80103528e-01,  1.17446816e+00],
 [-6.49185598e-01,  5.75075030e-01],
 [-1.44554257e+00, -1.20321488e+00],
 [ 6.42676830e-01, -1.32573023e-01],
 [-1.30800104e+00, -2.71790385e+00]],
```

```
[[-1.84748694e-02, -1.15813899e+00],
 [-1.00879085e+00, -1.03255343e+00],
 [ 1.58219028e+00,  1.38922203e+00],
 [ 3.38780075e-01, -4.95638967e-01],
 [ 9.21876907e-01,  1.21221922e-01],
 [-6.82340324e-01,  8.60912859e-01],
 [ 7.69151598e-02, -5.85590005e-01],
 [ 5.14049113e-01, -1.04417861e+00]],

[[-1.70082951e+00, -8.89654815e-01],
 [ 6.59987509e-01, -1.07014048e+00],
 [ 9.59146738e-01, -9.54873264e-01],
 [ 1.31178999e+00, -1.15762556e+00],
 [-8.80768597e-01, -1.00471556e+00],
 [-7.88973093e-01, -2.28176713e+00],
 [-2.11600995e+00,  6.60909235e-01],
 [ 9.83509600e-01,  5.48357129e-01]],


[[-8.89609039e-01,  8.91783893e-01],
 [ 5.49479485e-01,  9.99723792e-01],
 [ 4.41235900e-01, -3.90489399e-01],
 [-1.08281398e+00,  3.31635386e-01],
 [ 9.54425335e-01,  1.07885385e+00],
 [ 3.29850578e+00,  1.42687964e+00],
 [-1.51275754e+00, -3.21622372e-01],
 [ 1.04190099e+00, -1.19927347e+00]],

[[-7.63650060e-01, -4.08858746e-01],
 [ 1.73840165e+00, -7.96201468e-01],
 [ 1.17046297e+00,  1.59934759e+00],
 [-6.90645635e-01, -9.10946354e-02],
 [-5.58866322e-01,  5.70397019e-01],
 [ 9.46135879e-01, -1.40835536e+00],
 [-8.92387688e-01, -1.45307362e+00],
 [-9.77172613e-01,  1.92501649e-01]],

[[-1.20086503e+00, -8.10193419e-01],
 [ 1.23803926e+00, -8.10556412e-01],
 [-1.08264422e+00,  9.35884237e-01],
 [ 1.09055674e+00, -3.33833218e-01],
 [-1.42518267e-01,  8.64728987e-01],
 [-1.46374476e+00, -1.30505979e-01],
 [-1.45393282e-01,  9.04863104e-02],
 [ 1.68528911e-02, -1.06776142e+00]],

[[-1.31734192e+00, -8.82103682e-01],
 [ 1.06104684e+00,  2.91297108e-01],
 [ 2.48185086e+00,  1.19215131e+00],
 [ 1.92832124e+00,  1.54765272e+00],
```

xxx

```
       [ 6.70526683e-01, -2.93228656e-01],
       [-6.59248590e-01,  3.36534858e-01],
       [-9.14488316e-01, -2.89273024e-01],
       [ 5.42941093e-01, -2.89387465e-01]]], dtype=float32)>
```

```
a=tf.random.normal([3,8])
b=tf.random.normal([3,8])
tf.stack([a,b],axis=0)
```

Output:

```
<tf.Tensor: shape=(2, 3, 8), dtype=float32, numpy=
array([[[ 0.17763062,  0.24469708, -2.5346153 , -0.18690298,
          1.1443461 , -0.5390909 ,  0.16887239, -0.83570236],
        [ 1.8441521 ,  1.5235739 , -0.64010423, -0.37595153,
         -1.2121222 ,  0.10155217, -0.24727689,  0.5601055 ],
        [ 2.0012398 ,  0.669535  , -0.31648597,  0.14202364,
         -0.45610768,  0.9802554 ,  1.2451171 , -0.06139581]],

       [[-0.9079501 , -1.7644553 , -2.0430703 ,  0.2908407 ,
         -0.3038279 , -0.57020944,  0.18257792,  0.8421126 ],
        [-0.99999154,  0.6979242 ,  0.7644235 , -1.0942221 ,
         -1.0517521 , -1.4303746 , -1.4730486 ,  0.83887905],
        [-0.78506166, -0.9530724 , -0.37273416,  0.24031608,
          0.6013235 , -0.23241769,  1.0620469 , -1.044994  ]]],
      dtype=float32)>
```

```
x=tf.random.normal([10,35,8])
x
```

Output:

```
<tf.Tensor: shape=(10, 35, 8), dtype=float32, numpy=
array([[[ 1.247428  ,  1.0645797 ,  1.0592501 , ..., -1.961655  ,
          0.10052346,  1.2760571 ],
        [ 1.2604992 , -0.15541121,  0.58920753, ..., -1.0150295 ,
          1.7702786 ,  0.3993865 ],
        [ 1.5556142 ,  0.08028774, -0.04643722, ..., -0.07943559,
         -0.6282444 ,  1.9243791 ],
        ...,
        [-0.6491927 , -1.638288  ,  1.0260775 , ...,  1.9992698 ,
          2.6008942 ,  0.07995987],
        [-0.49557   ,  3.5592475 ,  1.5085462 , ...,  1.5545073 ,
         -0.97932374, -0.9667512 ],
        [ 0.1934562 ,  0.4828146 , -0.7754576 , ...,  0.6359363 ,
         -0.14738052,  1.1531788 ]],

       [[-0.28953525, -1.3332375 ,  0.24925381, ..., -0.31425712,
          1.6979342 ,  1.1464568 ],
        [-0.04866125, -0.55211556,  0.28995198, ..., -0.54563355,
          0.8838189 , -0.29077435],
```

```
 [ 0.43374658, -0.26102716, -0.91795325, ...,  1.091041  ,
   0.30386192, -1.4290375 ],
 ...,
 [ 0.759849  , -1.6247051 , -0.2584757 , ...,  0.04967271,
   0.28772992,  0.35292193],
 [-0.85510683, -0.65586233,  1.103383  , ..., -0.0373552 ,
   0.79138374, -0.5024714 ],
 [ 1.4885801 , -1.0929224 ,  0.07314687, ...,  0.66404504,
  -1.2330446 ,  1.2716359 ]],

[[-1.2873996 , -1.3831708 ,  0.0453666 , ...,  2.205461  ,
  -0.5195605 , -0.6754985 ],
 [ 0.39150372, -0.4558758 , -0.6098005 , ..., -0.02539941,
  -0.39720973, -0.43424475],
 [ 0.4112559 , -1.1688998 , -1.1747679 , ..., -0.42234674,
   0.21514174,  1.509846  ],
 ...,
 [-0.36276463, -0.11971979, -0.9530498 , ..., -0.19083904,
   0.18208185, -0.14189954],
 [ 1.2776968 , -0.26531413,  1.7855258 , ...,  0.02883373,
  -0.48796573, -0.39848393],
 [ 0.02551666,  1.6876936 , -0.36859158, ..., -0.91565686,
   0.7875778 , -0.1079988 ]],

...,

[[ 0.5652993 ,  0.63625026, -0.8600286 , ..., -0.2616015 ,
  -1.8072536 , -1.140475  ],
 [ 0.4365903 , -0.26730037,  0.3479633 , ..., -1.3586533 ,
   0.5778877 ,  0.420898  ],
 [-0.87385803, -1.410662  ,  1.0143865 , ..., -1.7114536 ,
   0.5397813 , -0.06425162],
 ...,
 [ 0.12660037,  1.9247079 ,  0.02370712, ..., -2.293575  ,
  -1.2561666 ,  0.40112025],
 [ 1.277536  , -1.2328033 ,  0.45963246, ...,  0.38388258,
  -0.2780825 , -0.6808684 ],
 [ 0.47438055, -1.1117886 , -1.0262231 , ..., -0.59383476,
  -0.5590973 , -0.4595411 ]],

[[ 0.19136156,  0.43603852, -0.4981979 , ..., -1.0880585 ,
   1.1172704 ,  1.394351  ],
 [-1.4701511 , -1.4711933 , -2.132695  , ..., -0.1370466 ,
   1.4501333 ,  0.24601784],
 [ 1.1397768 ,  0.50146174,  1.716779  , ..., -0.81130946,
  -1.318794  ,  0.22326742],
 ...,
 [-0.22040966, -1.593249  , -0.6774236 , ...,  1.3441583 ,
  -1.4486544 , -0.43220514],
 [-0.4615505 , -0.29952925, -1.3690901 , ..., -0.8109199 ,
```

```
         0.04641702, -0.43840653],
        [-0.48028305,  0.6266586 , -1.0863459 , ..., -1.3488654 ,
         1.1700169 ,  0.21116593]],

       [[ 0.09858859, -0.39052457,  0.6731252 , ..., -2.4032333 ,
         0.02606021, -0.96614975],
        [-0.44538796, -0.14375097, -1.9709284 , ..., -0.9896477 ,
         0.28646383,  0.37238717],
        [-0.3663308 , -0.30126634, -1.7911222 , ...,  0.46364367,
         0.34440812,  1.422815  ],
        ...,
        [ 0.2399586 ,  0.5894951 ,  0.28325137, ..., -0.9763484 ,
        -0.06056108,  0.68766296],
        [ 0.24392459,  0.25449648, -1.897226  , ...,  1.0934204 ,
         0.1331119 , -0.69305325],
        [-0.42335957, -1.9733636 , -0.822611  , ..., -0.14751941,
        -0.9815357 ,  0.45427302]]], dtype=float32)>
```

```
result=tf.split(x,num_or_size_splits=2,axis=0)
result
```

Output:

```
[<tf.Tensor: shape=(5, 35, 8), dtype=float32, numpy=
 array([[[ 1.247428  ,  1.0645797 ,  1.0592501 , ..., -1.961655  ,
          0.10052346,  1.2760571 ],
        [ 1.2604992 , -0.15541121,  0.58920753, ..., -1.0150295 ,
          1.7702786 ,  0.3993865 ],
        [ 1.5556142 ,  0.08028774, -0.04643722, ..., -0.07943559,
         -0.6282444 ,  1.9243791 ],
        ...,
        [-0.6491927 , -1.638288  ,  1.0260775 , ...,  1.9992698 ,
          2.6008942 ,  0.07995987],
        [-0.49557   ,  3.5592475 ,  1.5085462 , ...,  1.5545073 ,
         -0.97932374, -0.9667512 ],
        [ 0.1934562 ,  0.4828146 , -0.7754576 , ...,  0.6359363 ,
         -0.14738052,  1.1531788 ]],

       [[-0.28953525, -1.3332375 ,  0.24925381, ..., -0.31425712,
          1.6979342 ,  1.1464568 ],
        [-0.04866125, -0.55211556,  0.28995198, ..., -0.54563355,
          0.8838189 , -0.29077435],
        [ 0.43374658, -0.26102716, -0.91795325, ...,  1.091041  ,
          0.30386192, -1.4290375 ],
        ...,
        [ 0.759849  , -1.6247051 , -0.2584757 , ...,  0.04967271,
          0.28772992,  0.35292193],
        [-0.85510683, -0.65586233,  1.103383  , ..., -0.0373552 ,
          0.79138374, -0.5024714 ],
        [ 1.4885801 , -1.0929224 ,  0.07314687, ...,  0.66404504,
         -1.2330446 ,  1.2716359 ]],
```

```
        [[-1.2873996 , -1.3831708 ,  0.0453666 , ...,  2.205461  ,
         -0.5195605 , -0.6754985 ],
        [ 0.39150372, -0.4558758 , -0.6098005 , ..., -0.02539941,
         -0.39720973, -0.43424475],
        [ 0.4112559 , -1.1688998 , -1.1747679 , ..., -0.42234674,
          0.21514174,  1.509846  ],
        ...,
        [-0.36276463, -0.11971979, -0.9530498 , ..., -0.19083904,
          0.18208185, -0.14189954],
        [ 1.2776968 , -0.26531413,  1.7855258 , ...,  0.02883373,
         -0.48796573, -0.39848393],
        [ 0.02551666,  1.6876936 , -0.36859158, ..., -0.91565686,
          0.7875778 , -0.1079988 ]],

       [[-0.04099244, -0.4641188 , -1.1884779 , ...,  0.45390177,
         -0.73850036, -1.375087  ],
        [-0.04294039, -0.1133687 ,  0.08815806, ...,  0.59646976,
          0.60694647,  0.06499155],
        [ 0.7118572 ,  0.5563379 ,  0.10994585, ...,  0.1503733 ,
         -1.5061545 , -0.8461084 ],
        ...,
        [ 1.4198005 ,  0.06632233,  0.92517275, ...,  0.82579345,
         -0.19402687, -0.637355  ],
        [ 1.5964272 ,  0.03169739,  0.06193615, ..., -1.609944  ,
         -1.5827326 , -0.5943421 ],
        [ 0.5478164 ,  0.46689218,  2.3325553 , ..., -0.22920239,
          1.1370257 , -0.21173458]],

       [[ 0.50070584, -0.4696447 ,  1.5596375 , ..., -0.96774584,
         -0.34600022,  0.27333447],
        [ 0.01446368, -0.6145114 , -0.4091601 , ..., -0.989354  ,
         -0.1118072 , -0.14263047],
        [ 0.9325349 ,  1.4246366 ,  0.32882702, ...,  1.5467153 ,
          0.27443808, -2.0259476 ],
        ...,
        [-0.35845497, -0.44193843, -0.09360848, ...,  1.8857869 ,
         -1.368535  ,  0.19364443],
        [-1.4843849 , -0.26262897, -0.66967285, ...,  0.05519794,
         -0.38900027,  1.6266165 ],
        [-1.8666857 ,  0.93656343, -0.63401806, ...,  0.7999781 ,
          0.4903851 ,  0.4500667 ]]], dtype=float32)>,
<tf.Tensor: shape=(5, 35, 8), dtype=float32, numpy=
array([[[-1.4902195 ,  0.4498927 , -0.2015547 , ..., -0.1666371 ,
          0.12424421, -0.05857591],
        [-0.38135973,  0.52581453,  0.417967  , ..., -0.6648716 ,
         -0.73514855,  1.5436621 ],
        [-1.4229308 ,  1.0896119 , -0.2080162 , ..., -0.8569262 ,
         -1.245061  , -0.34481472],
        ...,
```

xxx

[-0.98898   , -0.9161218 , 1.1343044 , ..., -0.11192973,
  -0.34872416, 0.18515727],
 [0.04289725, 0.59023863, -0.84239614, ..., -0.04161176,
   1.1182595 , 1.873603],
 [0.4941957 , 1.0002222 , 1.7742091 , ..., 0.35288465,
  -0.08651639, 0.63943744]],

[[-0.5186036 , 1.6653761 , 0.17459558, ..., -1.2829549 ,
  -1.2325668 , -0.3998357],
 [0.3678929 , -0.41221994, 1.0325496 , ..., 0.67377645,
  -0.03063336, -0.43123788],
 [-2.25183   , 1.4724797 , 0.85960007, ..., 0.3623678 ,
   0.6733263 , -1.3020229],
 ...,
 [-0.37600535, -0.39644608, 2.54037   , ..., -0.59567374,
  -2.0568607 , 0.53867924],
 [-0.49809834, 0.73671883, -1.0966597 , ..., 1.5248111 ,
  -1.503932 , -0.07762504],
 [3.2009618 , 1.0615784 , -0.41338614, ..., 2.1030903 ,
  -0.8878297 , -1.568082]],

[[0.5652993 , 0.63625026, -0.8600286 , ..., -0.2616015 ,
  -1.8072536 , -1.140475],
 [0.4365903 , -0.26730037, 0.3479633 , ..., -1.3586533 ,
   0.5778877 , 0.420898],
 [-0.87385803, -1.410662 , 1.0143865 , ..., -1.7114536 ,
   0.5397813 , -0.06425162],
 ...,
 [0.12660037, 1.9247079 , 0.02370712, ..., -2.293575 ,
  -1.2561666 , 0.40112025],
 [1.277536 , -1.2328033 , 0.45963246, ..., 0.38388258,
  -0.2780825 , -0.6808684],
 [0.47438055, -1.1117886 , -1.0262231 , ..., -0.59383476,
  -0.5590973 , -0.4595411]],

[[0.19136156, 0.43603852, -0.4981979 , ..., -1.0880585 ,
   1.1172704 , 1.394351],
 [-1.4701511 , -1.4711933 , -2.132695 , ..., -0.1370466 ,
   1.4501333 , 0.24601784],
 [1.1397768 , 0.50146174, 1.716779 , ..., -0.81130946,
  -1.318794 , 0.22326742],
 ...,
 [-0.22040966, -1.593249 , -0.6774236 , ..., 1.3441583 ,
  -1.4486544 , -0.43220514],
 [-0.4615505 , -0.29952925, -1.3690901 , ..., -0.8109199 ,
   0.04641702, -0.43840653],
 [-0.48028305, 0.6266586 , -1.0863459 , ..., -1.3488654 ,
   1.1700169 , 0.21116593]],

[[0.09858859, -0.39052457, 0.6731252 , ..., -2.4032333 ,

```
           0.02606021, -0.96614975],
         [-0.44538796, -0.14375097, -1.9709284 , ..., -0.9896477 ,
           0.28646383,  0.37238717],
         [-0.3663308 , -0.30126634, -1.7911222 , ...,  0.46364367,
           0.34440812,  1.422815  ],
         ...,
         [ 0.2399586 ,  0.5894951 ,  0.28325137, ..., -0.9763484 ,
          -0.06056108,  0.68766296],
         [ 0.24392459,  0.25449648, -1.897226  , ...,  1.0934204 ,
           0.1331119 , -0.69305325],
         [-0.42335957, -1.9733636 , -0.822611  , ..., -0.14751941,
          -0.9815357 ,  0.45427302]]], dtype=float32)>]
```

```
result=tf.split(x,num_or_size_splits=[4,2,2,2],axis=0)
print(result)
len(result)
```

Output:

```
[<tf.Tensor: shape=(4, 35, 8), dtype=float32, numpy=
array([[[ 1.247428  ,  1.0645797 ,  1.0592501 , ..., -1.961655  ,
          0.10052346,  1.2760571 ],
        [ 1.2604992 , -0.15541121,  0.58920753, ..., -1.0150295 ,
          1.7702786 ,  0.3993865 ],
        [ 1.5556142 ,  0.08028774, -0.04643722, ..., -0.07943559,
         -0.6282444 ,  1.9243791 ],
        ...,
        [-0.6491927 , -1.638288  ,  1.0260775 , ...,  1.9992698 ,
          2.6008942 ,  0.07995987],
        [-0.49557   ,  3.5592475 ,  1.5085462 , ...,  1.5545073 ,
         -0.97932374, -0.9667512 ],
        [ 0.1934562 ,  0.4828146 , -0.7754576 , ...,  0.6359363 ,
         -0.14738052,  1.1531788 ]],

       [[-0.28953525, -1.3332375 ,  0.24925381, ..., -0.31425712,
          1.6979342 ,  1.1464568 ],
        [-0.04866125, -0.55211556,  0.28995198, ..., -0.54563355,
          0.8838189 , -0.29077435],
        [ 0.43374658, -0.26102716, -0.91795325, ...,  1.091041  ,
          0.30386192, -1.4290375 ],
        ...,
        [ 0.759849  , -1.6247051 , -0.2584757 , ...,  0.04967271,
          0.28772992,  0.35292193],
        [-0.85510683, -0.65586233,  1.103383  , ..., -0.0373552 ,
          0.79138374, -0.5024714 ],
        [ 1.4885801 , -1.0929224 ,  0.07314687, ...,  0.66404504,
         -1.2330446 ,  1.2716359 ]],
```

xxx

```
       [[-1.2873996 , -1.3831708 ,  0.0453666 , ...,  2.205461  ,
         -0.5195605 , -0.6754985 ],
        [ 0.39150372, -0.4558758 , -0.6098005 , ..., -0.02539941,
```

```
        -0.39720973, -0.43424475],
       [ 0.4112559 , -1.1688998 , -1.1747679 , ..., -0.42234674,
         0.21514174,  1.509846  ],
       ...,
       [-0.36276463, -0.11971979, -0.9530498 , ..., -0.19083904,
         0.18208185, -0.14189954],
       [ 1.2776968 , -0.26531413,  1.7855258 , ...,  0.02883373,
        -0.48796573, -0.39848393],
       [ 0.02551666,  1.6876936 , -0.36859158, ..., -0.91565686,
         0.7875778 , -0.1079988 ]],

      [[-0.04099244, -0.4641188 , -1.1884779 , ...,  0.45390177,
        -0.73850036, -1.375087  ],
       [-0.04294039, -0.1133687 ,  0.08815806, ...,  0.59646976,
         0.60694647,  0.06499155],
       [ 0.7118572 ,  0.5563379 ,  0.10994585, ...,  0.1503733 ,
        -1.5061545 , -0.8461084 ],
       ...,
       [ 1.4198005 ,  0.06632233,  0.92517275, ...,  0.82579345,
        -0.19402687, -0.637355  ],
       [ 1.5964272 ,  0.03169739,  0.06193615, ..., -1.609944  ,
        -1.5827326 , -0.5943421 ],
       [ 0.5478164 ,  0.46689218,  2.3325553 , ..., -0.22920239,
         1.1370257 , -0.21173458]]]], dtype=float32)>, <tf.Tensor: shape=(2, 35, 8), dtype=float32,
numpy=
array([[[ 5.00705838e-01, -4.69644696e-01,  1.55963755e+00,
         1.60837817e+00,  5.90789378e-01, -9.67745841e-01,
        -3.46000224e-01,  2.73334473e-01],
       [ 1.44636836e-02, -6.14511371e-01, -4.09160107e-01,
        -1.09814930e+00, -7.40380213e-03, -9.89354014e-01,
        -1.11807205e-01, -1.42630473e-01],
       [ 9.32534873e-01,  1.42463660e+00,  3.28827024e-01,
         2.72089154e-01, -8.12091172e-01,  1.54671526e+00,
         2.74438083e-01, -2.02594757e+00],
       [-1.20137644e+00,  7.04716265e-01, -5.38728893e-01,
        -9.23298717e-01,  1.35664213e+00,  1.58947563e+00,
        -4.63391021e-02,  5.27711928e-01],
       [-1.26721931e+00,  2.81842738e-01, -1.18687165e+00,
         9.25262719e-02,  6.92999125e-01,  1.15412807e+00,
        -8.31987917e-01,  1.18316674e+00],
       [-1.55338657e+00,  4.40612584e-02,  5.57216287e-01,
        -1.80529594e+00,  4.26462233e-01, -3.32232445e-01,
         8.82961035e-01,  6.25158727e-01],
       [-5.96968412e-01, -2.42800474e+00, -2.54927278e-01,
         1.24198623e-01, -4.60400134e-01, -1.25588405e+00,
         5.48572719e-01,  3.04215848e-01],
       [-1.58074692e-01, -1.74294531e+00,  8.24439302e-02,
         1.65964258e+00, -1.40721941e+00, -1.29874134e+00,
        -8.77593219e-01,  1.42013621e+00],
       [ 2.02747893e+00, -3.98356199e-01, -1.26070499e+00,
```

```
 7.66003609e-01, -5.12763262e-01,  5.02238810e-01,
  1.23382103e+00,  7.53484130e-01],
[-1.91660869e+00, -3.30009639e-01, -5.04793704e-01,
 -7.58954763e-01,  1.11190766e-01,  1.69295239e+00,
  1.84559631e+00,  1.06217198e-01],
[ 1.19259286e+00,  6.52127504e-01,  3.25928926e+00,
 -1.62294898e-02, -4.27431226e-01, -1.18566489e+00,
 -7.52523124e-01,  1.74056981e-02],
[ 1.07795000e+00, -1.75349295e+00, -1.71702611e+00,
  2.47858930e+00,  8.60156655e-01,  6.28931165e-01,
 -7.92876065e-01, -1.01485419e+00],
[ 2.25729167e-01,  1.95653081e+00,  8.35446775e-01,
  4.33610007e-02,  2.33570546e-01, -6.05315864e-01,
 -7.21947372e-01,  6.46357015e-02],
[ 1.32646179e+00,  1.63309538e+00,  6.30227506e-01,
 -2.84485906e-01,  1.61828339e+00, -1.27560711e+00,
  5.66022933e-01,  2.26124215e+00],
[ 4.31685328e-01, -7.15706944e-01, -1.30592358e+00,
 -5.25248460e-02, -8.23542595e-01, -1.06788301e+00,
 -9.47785795e-01, -1.92150891e-01],
[ 1.39866889e+00,  1.44662678e+00, -8.17103088e-02,
  1.02189982e+00,  1.81123659e-01,  8.06939363e-01,
  8.65728080e-01, -1.17518544e-01],
[-3.81735891e-01, -9.13468659e-01, -6.98263824e-01,
  1.98594317e-01, -8.69992793e-01,  1.56490064e+00,
 -1.12887132e+00, -1.49158227e+00],
[ 5.53670466e-01,  1.33353889e+00, -3.58389497e-01,
  1.29526401e+00,  1.95643485e+00,  1.52892101e+00,
 -1.54378939e+00,  6.31635904e-01],
[-1.12630093e+00,  7.62899756e-01,  1.39434135e+00,
  2.15476632e+00, -1.40685034e+00, -4.29717898e-01,
 -8.72992337e-01, -4.24088955e-01],
[ 1.83798134e+00,  6.79894269e-01,  9.22676861e-01,
  1.18162763e+00,  7.08951414e-01, -8.00180793e-01,
 -1.54797271e-01, -1.11664617e+00],
[-3.62391740e-01,  7.53948390e-01,  1.24944818e+00,
 -8.97169352e-01,  1.11915290e+00, -5.46539783e-01,
 -1.16539560e-01,  1.18284583e+00],
[ 1.78211525e-01,  1.94682121e+00,  7.33111799e-01,
 -1.03078198e+00,  3.53112429e-01,  3.90855610e-01,
 -3.26376945e-01, -1.81484950e+00],
[-2.16850236e-01, -7.04559386e-01,  2.10996509e+00,
 -1.55047625e-01, -5.55356503e-01,  1.46145153e+00,
  9.59583938e-01, -2.02495202e-01],
[-8.92413259e-01, -2.06607342e-01, -6.90058023e-02,
  1.69172734e-01,  1.03505719e+00,  1.63502824e-02,
 -4.46272016e-01,  1.60512149e-01],
[ 8.12018663e-02, -6.25445470e-02, -1.21611707e-01,
  9.44693625e-01, -1.12167299e+00,  5.52691281e-01,
  5.17091274e-01, -4.32394058e-01],
```

```
[ 1.32433867e+00,  7.77884722e-01,  2.19110176e-01,
  1.56274009e+00, -8.16892385e-01, -5.68201721e-01,
 -1.10127307e-01, -4.94969666e-01],
[ 2.92204112e-01, -8.86786461e-01, -7.28188157e-01,
  9.47472394e-01, -8.00105989e-01,  8.92548636e-02,
 -9.82300282e-01,  1.19731498e+00],
[-2.37873629e-01, -1.46714723e+00,  2.18841505e+00,
 -1.00537956e+00,  5.16527355e-01, -6.96576536e-01,
 -6.01065814e-01,  8.50466251e-01],
[ 1.59001303e+00, -7.52438605e-01,  1.03309226e+00,
  4.58909959e-01, -1.25273108e+00, -1.01127648e+00,
  1.40936172e+00,  3.36813241e-01],
[ 1.98332083e+00, -1.25232661e+00,  6.49978280e-01,
  2.32810657e-02,  4.37820524e-01, -1.42785162e-01,
  5.88211358e-01, -1.11682177e+00],
[ 3.83515507e-01, -4.36417818e-01, -1.80875994e-02,
  1.51901054e+00,  4.19627488e-01, -9.88368034e-01,
  2.25688532e-01,  9.15946007e-01],
[-1.09222281e+00,  1.09104192e+00, -1.81924891e+00,
  4.88837868e-01,  1.00426450e-01,  8.52906585e-01,
  2.05498481e+00, -2.99179941e-01],
[-3.58454973e-01, -4.41938430e-01, -9.36084837e-02,
  2.63915569e-01,  8.18441153e-01,  1.88578689e+00,
 -1.36853504e+00,  1.93644434e-01],
[-1.48438489e+00, -2.62628973e-01, -6.69672847e-01,
 -7.35318303e-01, -1.87138987e+00,  5.51979393e-02,

  1.88331604e-01, -8.76994491e-01,  1.08056581e+00,
  3.35453600e-01,  5.61376929e-01],
[-2.46224809e+00,  1.04573406e-01,  7.67947555e-01,
  4.45797265e-01,  1.81032228e+00, -3.56285393e-01,
 -9.40442324e-01, -1.20358551e+00],
[ 1.21311176e+00, -2.68113941e-01, -1.40964687e+00,
  9.79326144e-02, -1.16712916e+00, -3.84339958e-01,
  6.37127236e-02, -1.33854735e+00],
[-1.25337660e+00,  4.29824173e-01,  6.19965672e-01,
  4.83034462e-01,  2.53634065e-01, -1.41655457e+00,
 -2.61087447e-01,  6.11919165e-02],
[ 4.23986930e-03, -1.18687220e-01,  6.03586853e-01,
  8.15088570e-01,  1.36938322e+00,  5.17867684e-01,
  6.79204583e-01, -7.71912098e-01],
[-3.71907592e-01,  2.89772391e-01, -1.46614552e-01,
  3.28848064e-01,  1.18073380e+00,  7.08369493e-01,
  8.54159713e-01, -1.25956297e+00],
[-5.82941175e-01,  1.12861645e+00, -3.61305445e-01,
 -2.23183677e-01, -6.16805077e-01,  1.01571321e+00,
 -3.94261479e-01,  6.17441475e-01],
[ 2.33007044e-01,  2.53309458e-01,  1.29175745e-02,
  2.06585839e-01,  1.28172076e+00,  1.01903714e-01,
```

```
         1.10403442e+00, -1.62413493e-01],
       [-5.83879650e-01, -1.57087123e+00,  9.29846525e-01,
         1.08420396e+00,  4.21488732e-01, -7.60115862e-01,
        -3.59579414e-01,  7.35322773e-01],
       [ 2.84206063e-01,  2.52417564e+00,  9.95647192e-01,
         3.80149961e-01, -1.02508888e-02,  1.02789211e+00,
        -7.30230689e-01,  4.10768241e-01],
       [ 2.39958599e-01,  5.89495122e-01,  2.83251375e-01,
        -1.28155211e-02,  4.99131978e-01, -9.76348400e-01,
        -6.05610758e-02,  6.87662959e-01],
       [ 2.43924588e-01,  2.54496485e-01, -1.89722598e+00,
        -2.01582766e+00,  7.72778273e-01,  1.09342039e+00,
         1.33111894e-01, -6.93053246e-01],
       [-4.23359573e-01, -1.97336364e+00, -8.22610974e-01,
        -2.82488406e-01,  3.75018179e-01, -1.47519410e-01,
        -9.81535673e-01,  4.54273015e-01]]], dtype=float32)>]
```

```
4
```

```
a=tf.ones([2,2])
tf.norm(x,ord=1)
```

Output:

```
<tf.Tensor: shape=(), dtype=float32, numpy=2234.3909>
```

```
tf.norm(x,ord=2)
```

Output:

```
<tf.Tensor: shape=(), dtype=float32, numpy=52.906647>
```

```
import numpy as np
tf.norm(x,ord=np.inf)
```

Output:

```
<tf.Tensor: shape=(), dtype=float32, numpy=3.5592475>
```

```
x=tf.random.normal([4,10])
x
```

Output:

```
<tf.Tensor: shape=(4, 10), dtype=float32, numpy=
array([[-0.8842504 , -0.5509507 , -0.7801942 ,  1.1252639 , -0.19370236,
        -1.9772154 , -0.5801756 , -0.9374427 , -0.20289214, -0.23877631],
       [ 1.283532  , -0.57703197, -0.5883134 , -0.11461499,  1.6858338 ,
         1.001978  , -1.6884319 , -0.64083254, -0.6094569 ,  0.74931633],
       [-0.7950863 , -1.2719189 ,  0.11113866,  0.40813583,  0.22533345,
        -0.66343164,  0.02669905, -0.28859156, -0.8679272 ,  0.29442388],
       [-0.69701284, -0.7496005 , -0.19896737,  1.5073245 ,  1.3989887 ,
```

```
  -2.344581  , -0.7622699 , -0.5573321 , -0.4940968 ,  1.5422927 ]],
 dtype=float32)>
```

```
tf.reduce_max(x,axis=0)
```

Output:

```
<tf.Tensor: shape=(10,), dtype=float32, numpy=
array([ 1.283532  , -0.5509507 ,  0.11113866,  1.5073245 ,  1.6858338 ,
        1.001978  ,  0.02669905, -0.28859156, -0.20289214,  1.5422927 ],
      dtype=float32)>
```

```
tf.reduce_max(x,axis=1)
```

Output:

```
<tf.Tensor: shape=(4,), dtype=float32, numpy=array([1.1252639 , 1.6858338 , 0.40813583, 1.5422927 ], dtype=float32)>
```

```
tf.reduce_min(x,axis=0)
```

Output:

```
<tf.Tensor: shape=(10,), dtype=float32, numpy=
array([-0.8842504 , -1.2719189 , -0.7801942 , -0.11461499, -0.19370236,
       -2.344581  , -1.6884319 , -0.9374427 , -0.8679272 , -0.23877631],
      dtype=float32)>
```

```
tf.reduce_min(x,axis=1)
```

Output:

```
<tf.Tensor: shape=(4,), dtype=float32, numpy=array([-1.9772154, -1.6884319, -1.2719189, -2.344581 ], dtype=float32)>
```

Platform used : Kaggle

Conclusion:

Created tensors and performed merge,split and statistical operations.

Experiment – 4

Aim: Design a single unit perceptron for the classification of the Iris dataset without using pre-defined models.

Context: The domain for this classifier can be Agriculture.The iris flower set involved by Fisher. The use of multiple measurements in taxonomic problems. It is also sometimes called Anderson's dataset because Anderson collected the data to classify the flowers.

Problem Statement: Iris flower dataset for multi-class classifier.

Data Set: The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of multiple measurements in taxonomic problems, can also be found on the UCI Machine Learning Repository.
It includes the iris species with 50 samples each, as well as some properties about each flower.One flower species is linearly separable from the other two,but the other two are not linearly separable.
The columns in this dataset are:
→Id
→Sepal Length CM
→Sepal Width CM
→Petal Length CM
→Petal Width CM
→Species
The species columns consist of 50 instances each of setosa,versicolour and veriginica.

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
iris = load_iris()
X = iris.data[:100, :2]
y = iris.target[:100]
y = np.where(y == 0, -1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2, random_state=42)
weights = np.zeros(X_train.shape[1])
bias = 0
learning_rate = 0.1
epochs = 10
for epoch in range(epochs):
    for i in range(X_train.shape[0]):
        linear_output = np.dot(X_train[i], weights) + bias
        y_pred = np.where(linear_output > 0, 1, -1)
        if y_train[i] != y_pred:
```

```
            weights += learning_rate * y_train[i] * X_train[i]
            bias += learning_rate * y_train[i]
X=iris.data[np.concatenate((np.arange(50),np.arange(100,50)))]
y=iris.target[np.concatenate((np.arange(50),np.arange(100,50)))]
y=np.where(y==0,-1,1)
correct_predictions = 0
for i in range(X_test.shape[0]):
    linear_output = np.dot(X_test[i], weights) + bias
    y_pred = np.where(linear_output > 0, 1, -1)
    if y_pred == y_test[i]:
        correct_predictions += 1
accuracy = correct_predictions / X_test.shape[0]
print(f"Accuracy: {accuracy * 100:.2f}%")
```

Output: Accuracy: 100.00%

Platform used: Kaggle

Conclusion: A single unit perceptron for the classification of the Iris dataset is done

Experiment - 5

Aim: Design, train, and test the MLP for tabular data and verify various activation functions and optimizers in TensorFlow.

Context: The domain for this classifier can be Agriculture.The iris flower set involved by Fisher. The use of multiple measurements in taxonomic problems. It is also sometimes called Anderson's dataset because Anderson collected the data to classify the flowers.

Problem Statement: Iris flower dataset for multi-class classifier.

Data Set: The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of multiple measurements in taxonomic problems, can also be found on the UCI Machine Learning Repository.
It includes the iris species with 50 samples each, as well as some properties about each flower.One flower species is linearly separable from the other two,but the other two are not linearly separable.
The columns in this dataset are:
→Id
→Sepal Length CM
→Sepal Width CM
→Petal Length CM
→Petal Width CM
→Species
The species columns consist of 50 instances each of setosa,versicolour and veriginica.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import SGD, Adam, RMSprop
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
import warnings
warnings.filterwarnings("ignore")

data= load_iris()
x=data.data
y=data.target

encoder= OneHotEncoder(sparse=False)
y=encoder.fit_transform(y.reshape(-1,1))

scaler= StandardScaler()
x=scaler.fit_transform(x)
```



```
x_train,x_test, y_train,y_test = train_test_split(x,y,test_size=0.2, random_state=42)
```

```python
def create_model(activation_func, optimizer):
  model= Sequential([
    Dense(64, input_dim=x_train.shape[1], activation=activation_func),Dropout(0.5),
    Dense(32,activation=activation_func),Dropout(0.5),
    Dense(3, activation = 'softmax')])
  model.compile(loss='categorical_crossentropy', optimizer= optimizer,metrics=['accuracy'])
  return model

activation_funcs = ['relu', 'sigmoid', 'tanh']


for activation_func in activation_funcs:
  # Create new optimizer instances inside the loop
  optimizers = [SGD(learning_rate=0.01), Adam(learning_rate=0.001),
RMSprop(learning_rate=0.001)]
  for optimizer in optimizers:
    model = create_model(activation_func, optimizer)
    model.fit(x_train, y_train, epochs=50, batch_size=16, verbose=0)
    loss, accuracy = model.evaluate(x_test, y_test, verbose=0)

    print(f'Activation: {activation_func}, Optimizer: {optimizer.__class__.__name__}, Loss:
{loss:.3f}, Accuracy: {accuracy:.3f}')
```

Output:

```
Activation: relu, Optimizer: SGD, Loss: 0.306, Accuracy: 0.933
Activation: relu, Optimizer: Adam, Loss: 0.142, Accuracy: 1.000
Activation: relu, Optimizer: RMSprop, Loss: 0.125, Accuracy: 0.967
Activation: sigmoid, Optimizer: SGD, Loss: 1.069, Accuracy: 0.633
Activation: sigmoid, Optimizer: Adam, Loss: 0.530, Accuracy: 0.900
Activation: sigmoid, Optimizer: RMSprop, Loss: 0.489, Accuracy: 0.900
Activation: tanh, Optimizer: SGD, Loss: 0.227, Accuracy: 0.967
Activation: tanh, Optimizer: Adam, Loss: 0.101, Accuracy: 1.000
Activation: tanh, Optimizer: RMSprop, Loss: 0.067, Accuracy: 1.000
```

Platform used: Kaggle

Scope for further improvement: The accuracy of the model can be improved by using predefined models such as VGGNet,ResNet,GoogleNet,MobileNet which have been already trained on the imagenet dataset. We can further add layers such as Batch Normalization or Dropout or more Dense layers to enhance its performance.

Conclusion: A multilayer perceptron model for tabular data has been constructed using various activation functions and optimizers.

Experiment - 6

Aim: Design and implement to classify 32x32 images using MLP with TensorFlow/Keras and check the accuracy.

Context: Shape Classification finds its use in many medical fields such as MRI Scans where the shape of the muscle is decided using this model

Problem Statement: This model classifies a 32X32 image into any of the three classes circles, squares or triangles.

Dataset: The dataset contains 3 folders with 100 images each of triangles, squares and circles. Each png image is 32x32 pixels which are distributed in 3 folders.

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import warnings
warnings.filterwarnings("ignore")
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
```

```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

```
y_test
```

Output:

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

```
model = Sequential()
model.add(Flatten(input_shape=(32, 32, 3)))
model.add(Dense(512, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
```



```
model.summary()
```

Output:

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 3072)	0
dense (Dense)	(None, 512)	1,573,376
dense_1 (Dense)	(None, 256)	131,328
dense_2 (Dense)	(None, 10)	2,570

Total params: 1,707,274 (6.51 MB)

Trainable params: 1,707,274 (6.51 MB)

Non-trainable params: 0 (0.00 B)

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
```

Output:

```
Epoch 1/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 22s 17ms/step - accuracy: 0.5737
- loss: 1.2006 - val_accuracy: 0.4764 - val_loss: 1.5569
Epoch 2/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 22s 18ms/step - accuracy: 0.5810
- loss: 1.1740 - val_accuracy: 0.4650 - val_loss: 1.6546
Epoch 3/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 22s 17ms/step - accuracy: 0.5799
- loss: 1.1785 - val_accuracy: 0.4865 - val_loss: 1.5297
Epoch 4/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.5867
- loss: 1.1499 - val_accuracy: 0.4815 - val_loss: 1.5980
Epoch 5/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.5902
- loss: 1.1438 - val_accuracy: 0.4883 - val_loss: 1.5432
Epoch 6/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.5891
- loss: 1.1377 - val_accuracy: 0.4846 - val_loss: 1.5664
Epoch 7/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.5943
- loss: 1.1322 - val_accuracy: 0.4733 - val_loss: 1.6064
Epoch 8/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.5953
- loss: 1.1336 - val_accuracy: 0.4778 - val_loss: 1.6055
```

```
Epoch 9/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.6004
- loss: 1.1193 - val_accuracy: 0.4724 - val_loss: 1.6151
Epoch 10/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 21s 17ms/step - accuracy: 0.5993
- loss: 1.1121 - val_accuracy: 0.4835 - val_loss: 1.5651
```

```
<keras.src.callbacks.history.History at 0x7bcb55f3d2d0>
```

```python
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_accuracy:.4f}')
```

Output:

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.4911 -
loss: 1.5433
Test accuracy: 0.4867
```

```python
model.save('mlp_cifar10_model.h5')
print("model saved")
```

```
model saved
```

Platform used: Kaggle

Scope for further improvement: Inorder to improve the performance of this model we can use models that have been already trained on image net dataset and have a good accuracy.

Conclusion: A multi-layer perceptron model is built to classify 32 * 32 images .

Experiment – 7

Aim: Design and implement a CNN model to classify multi-category JPG images with TensorFlow/Keras and check accuracy. Predict labels for new images

Problem Statement : Implementation of a CNN model

Dataset: Here we used the cifar dataset where we classify the multi category JPG images.

```
import warnings
warnings.filterwarnings("ignore")
import keras
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,  Dropout
from keras.datasets import cifar10
(x_train,y_train), (x_test , y_test)=cifar10.load_data()
```

```
%matplotlib inline
fig = plt.figure(figsize=(20,5))
for i in range(36):
   ax=fig.add_subplot(3,12,i+1,xticks=[], yticks=[])
   ax.imshow(np.squeeze(x_train[i]))
```

```
x_train=x_train.astype('float32')/255
x_test=x_test.astype('float32')/255
```

```
num_classes = len(np.unique(y_train))
y_train =keras.utils.to_categorical(y_train, num_classes)
y_test =keras.utils.to_categorical(y_test, num_classes)
```

```
(x_train, x_valid) = x_train[5000:], x_train[:5000]
(y_train, y_valid) = y_train[5000:], y_train[:5000]
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
print(x_valid.shape[0], 'validation samples')
```

```
model = Sequential()
model.add(Conv2D(filters=16, kernel_size=2, padding='same',
activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=32, kernel_size=2, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
```

```
model.add(Conv2D(filters=64, kernel_size=2, padding='same',
activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))
model.summary()

model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
metrics=['accuracy'])

hist = model.fit(x_train, y_train, batch_size=32, epochs=5,validation_data=(x_valid,
y_valid),verbose=1, shuffle=True)

score = model.evaluate(x_test, y_test, verbose=0)
print('\n', 'Test accuracy:', score[1])
```

Output:

Test accuracy: 73.05%

Platform Used: Kaggle

Conclusion: Implementation of a CNN model for classifying JPG images is done

Experiment – 8

Aim: Design and implement a CNN model to classify multi-category TIFF images with TensorFlow/Keras and check the accuracy. Check whether your model is overfit/underfit/perfect fit and apply the techniques to avoid overfit and underfit like regularizers, dropouts, etc.

Problem Statement: To design a CNN model to classify tiff images

Dataset: Here we used the cifar dataset where we classify the multi category JPG images.

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from PIL import Image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
import warnings
warnings.filterwarnings('ignore')

train_dir = 'cifar10_tiff/train'
test_dir = 'cifar10_tiff/test'
os.makedirs(train_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Output:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```

```
class_labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

def save_images(images, labels, directory):
  for i, (image_array, label) in enumerate(zip(images, labels)):
    image = Image.fromarray(image_array)
    label_name = class_labels[int(label)]
    label_dir = os.path.join(directory, label_name)
    os.makedirs(label_dir, exist_ok=True)
    image_path = os.path.join(label_dir, f"{label_name}_{i}.tiff")
    image.save(image_path, format='TIFF')
```

```
save_images(x_train, y_train, train_dir)
save_images(x_test, y_test, test_dir)
print("Images have been successfully saved as .tiff files.")
```

Output:

Images have been successfully saved as .tiff files.

```
train_dir = 'cifar10_tiff/train'
test_dir = 'cifar10_tiff/test'

train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    validation_split=0.2
)

test_datagen = ImageDataGenerator(rescale=1.0/255.0)

train_data = train_datagen.flow_from_directory(
    directory=train_dir,
    target_size=(32, 32),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)
validation_data = train_datagen.flow_from_directory(
    directory=train_dir,
    target_size=(32, 32),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
test_data = test_datagen.flow_from_directory(
    directory=test_dir,
    target_size=(32, 32),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)
```

Output:

Found 40000 images belonging to 10 classes.
Found 10000 images belonging to 10 classes.
Found 10000 images belonging to 10 classes.


```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
```

```python
    Dropout(0.25),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),

    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')  # 10 classes for CIFAR-10
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Output:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
dropout (Dropout)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 128)	0
dropout_2 (Dropout)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 256)	131,328

```
| dropout_3 (Dropout)          | (None, 256)          |           0 |
├──────────────────────────────┼──────────────────────┼─────────────┤
| dense_1 (Dense)              | (None, 10)           |       2,570 |
└──────────────────────────────┴──────────────────────┴─────────────┘
```

Total params: 227,146 (887.29 KB)

Trainable params: 227,146 (887.29 KB)

Non-trainable params: 0 (0.00 B)

```
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
history =
model.fit(train_data,validation_data=validation_data,epochs=10,callbacks=[early_stopping])
```

Output:

```
Epoch 1/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 88s 69ms/step - accuracy: 0.2499
- loss: 1.9800 - val_accuracy: 0.4465 - val_loss: 1.5052
Epoch 2/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 85s 68ms/step - accuracy: 0.4385
- loss: 1.5225 - val_accuracy: 0.5192 - val_loss: 1.3407
Epoch 3/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 84s 67ms/step - accuracy: 0.4908
- loss: 1.4101 - val_accuracy: 0.5493 - val_loss: 1.2594
Epoch 4/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 85s 67ms/step - accuracy: 0.5212
- loss: 1.3500 - val_accuracy: 0.5814 - val_loss: 1.1913
Epoch 5/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 85s 68ms/step - accuracy: 0.5393
- loss: 1.2920 - val_accuracy: 0.6048 - val_loss: 1.1288
Epoch 6/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 86s 68ms/step - accuracy: 0.5487
- loss: 1.2690 - val_accuracy: 0.6146 - val_loss: 1.1195
Epoch 7/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 88s 70ms/step - accuracy: 0.5645
- loss: 1.2209 - val_accuracy: 0.6093 - val_loss: 1.0969
Epoch 8/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 89s 71ms/step - accuracy: 0.5734
- loss: 1.2141 - val_accuracy: 0.5834 - val_loss: 1.1658
Epoch 9/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 87s 70ms/step - accuracy: 0.5814
- loss: 1.1832 - val_accuracy: 0.6319 - val_loss: 1.0336
Epoch 10/10
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 87s 69ms/step - accuracy: 0.5886
- loss: 1.1679 - val_accuracy: 0.6172 - val_loss: 1.0893
```



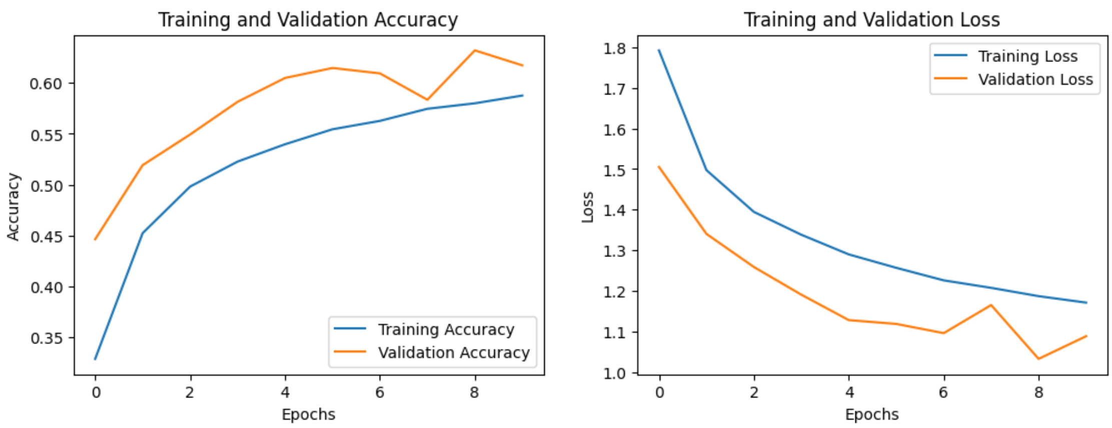
```
test_loss, test_accuracy = model.evaluate(test_data)
print(f"Test accuracy: {test_accuracy * 100:.2f}%")
```

Output:

313/313 ━━━━━━━━━━━━━━━━━━━━ 8s 25ms/step - accuracy: 0.6281 - loss: 1.0476
Test accuracy: 66.39%

```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training and Validation Loss')
plt.show()
```

Output:




```
predictions = model.predict(test_data)
predicted_classes = tf.argmax(predictions, axis=1)
```

Output:

313/313 ━━━━━━━━━━━━━━━━━━━━ 8s 25ms/step

```
true_classes = test_data.classes

accuracy = np.mean(predicted_classes == true_classes)
print(f"Prediction accuracy on test set: {accuracy * 100:.2f}%")
```

Output:

Prediction accuracy on test set: 66.39%

Platform used : Kaggle

Optimizer Used: Adam

Conclusion:

A CNN model has been constructed to classify tiff files and predict labels for new images.

Experiment - 9

Implement CNN architectures (LeNet, AlexNet, VGG, etc.) models to classify multi-category satellite images with TensorFlow/Keras and check the accuracy. Check whether your model is overfit/underfit/perfect fit and apply the techniques to avoid overfit and underfit.

AIM:

1. To implement and train different Convolutional Neural Network (CNN) architectures:

   a) LeNet-5
   b) AlexNet
   c) VGG-like network

2. To classify multi-category images from the **CIFAR-10 dataset**.
3. To compare the performance of the architectures based on validation accuracy.
4. To check for overfitting or underfitting using training and validation curves.

DATASET: CIFAR-10

- CIFAR-10 is a dataset of 60,000 images across 10 classes (e.g., airplanes, cars, cats, dogs, etc.).
- Each image is of size **32x32 pixels with 3 color channels (RGB)**.

REQUIREMENTS

- Python 3.x/Jupyter/Google Colab
- TensorFlow/Keras
- Matplotlib

PROCEDURE

Step 1: Load and Preprocess the Dataset

1. Import the CIFAR-10 dataset using TensorFlow/Keras.
2. Normalize the image data to scale pixel values between [0, 1].
3. One-hot encode the labels for classification.

Step 2: Define the CNN Architectures

1. Implement **LeNet-5** using two convolutional layers and dense layers.

2. Implement **AlexNet-like architecture** with multiple convolutional and dropout layers.
3. Implement a **VGG-like architecture** with deeper layers using 3x3 convolutions.

Step 3: Train and Evaluate the Models

1. Train each model for 10 epochs using the **Adam optimizer** and **categorical cross-entropy loss**.
2. Evaluate the models using validation accuracy on the test set.

Step 4: Compare the Models

1. Plot the **training and validation accuracy** of all models over epochs.
Compare test accuracy and identify the best-performing model.

Code:

```
# Import Required Libraries

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to_categorical

import matplotlib.pyplot as plt

# STEP 1: LOAD AND PREPROCESS DATA

# Load CIFAR-10 dataset

(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the pixel values to [0, 1]

x_train = x_train / 255.0

x_test = x_test / 255.0

# One-hot encode labels

y_train = to_categorical(y_train, 10)

y_test = to_categorical(y_test, 10)

# STEP 2: DEFINE CNN MODELS

# LeNet-5 Model

def lenet_model():

    model = Sequential([

        Conv2D(6, kernel_size=(5,5), activation='relu', input_shape=(32, 32, 3), padding='same'),
```

```
        MaxPooling2D(pool_size=(2,2)),
        Conv2D(16, kernel_size=(5,5), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2,2)),
        Flatten(),
        Dense(120, activation='relu'),
        Dense(84, activation='relu'),
        Dense(10, activation='softmax') # 10 classes
    ])
    return model
# AlexNet-like Model
def alexnet_model():
    model = Sequential([
        Conv2D(96, kernel_size=(3,3), strides=(1,1), activation='relu', input_shape=(32, 32, 3),
            padding='same'),
        MaxPooling2D(pool_size=(2,2)),
        Conv2D(256, kernel_size=(3,3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2,2)),
        Conv2D(384, kernel_size=(3,3), activation='relu', padding='same'),
        Conv2D(384, kernel_size=(3,3), activation='relu', padding='same'),
        Conv2D(256, kernel_size=(3,3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2,2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
```

```
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(10, activation='softmax')
    ])
    return model

# VGG-like Model
def vgg_model():
    model = Sequential([
        Conv2D(64, kernel_size=(3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
        Conv2D(64, kernel_size=(3,3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2,2)),
        Conv2D(128, kernel_size=(3,3), activation='relu', padding='same'),
        Conv2D(128, kernel_size=(3,3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2,2)),
        Conv2D(256, kernel_size=(3,3), activation='relu', padding='same'),
        Conv2D(256, kernel_size=(3,3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2,2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(10, activation='softmax')
    ])
```

```
    return model
# STEP 3: TRAIN AND EVALUATE MODELS
def train_and_evaluate(model, model_name, epochs=10, batch_size=64):
    print(f"\nTraining {model_name}...\n")
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    history = model.fit(x_train, y_train, epochs=epochs, batch_size=batch_size,
                        validation_data=(x_test, y_test), verbose=1)
    test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
    print(f"{model_name} Test Accuracy: {test_acc * 100:.2f}%")
    return history, test_acc
# Train LeNet
lenet = lenet_model()
lenet_history, lenet_acc = train_and_evaluate(lenet, "LeNet")
# Train AlexNet
alexnet = alexnet_model()
alexnet_history, alexnet_acc = train_and_evaluate(alexnet, "AlexNet")
# Train VGG
vgg = vgg_model()
vgg_history, vgg_acc = train_and_evaluate(vgg, "VGG")
# Plot Training and Validation Accuracy
def plot_history(histories, labels):
    plt.figure(figsize=(10, 6))
    for history, label in zip(histories, labels):
        plt.plot(history.history['val_accuracy'], label=f'{label} Validation Accuracy')
```

```
    plt.title('Model Comparison - Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
# Compare All Models
plot_history(
    histories=[lenet_history, alexnet_history, vgg_history],
    labels=['LeNet', 'AlexNet', 'VGG']
)
# Print Summary of Results
print("\nSummary of Results:")
print(f"LeNet Accuracy: {lenet_acc * 100:.2f}%")
print(f"AlexNet Accuracy: {alexnet_acc * 100:.2f}%")
print(f"VGG Accuracy: {vgg_acc * 100:.2f}%")
# Check which model performs best
best_model = max([("LeNet", lenet_acc), ("AlexNet", alexnet_acc), ("VGG", vgg_acc)],
key=lambda x: x[1])
print(f"\nThe best performing model is {best_model[0]} with an accuracy of {best_model[1] *
100:.2f}%.")
```

Output:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 13s 0us/step
Training LeNet...
Epoch 1/10
```

782/782 ━━━━━━━━━━━━━━━━━━━━ 14s 12ms/step - accuracy: 0.3109 - loss: 1.8388 - val_accuracy: 0.5224 - val_loss: 1.3391

Epoch 2/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5245 - loss: 1.3317 - val_accuracy: 0.5576 - val_loss: 1.2342

Epoch 3/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5753 - loss: 1.1964 - val_accuracy: 0.6018 - val_loss: 1.1342

Epoch 4/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.6146 - loss: 1.0883 - val_accuracy: 0.6087 - val_loss: 1.1264

Epoch 5/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.6417 - loss: 1.0197 - val_accuracy: 0.6254 - val_loss: 1.0679

Epoch 6/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.6607 - loss: 0.9605 - val_accuracy: 0.6337 - val_loss: 1.0478

Epoch 7/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.6781 - loss: 0.9151 - val_accuracy: 0.6251 - val_loss: 1.0567

Epoch 8/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.6943 - loss: 0.8784 - val_accuracy: 0.6263 - val_loss: 1.1134

Epoch 9/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.7123 - loss: 0.8188 - val_accuracy: 0.6495 - val_loss: 1.0072

Epoch 10/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.7237 - loss: 0.7839 - val_accuracy: 0.6537 - val_loss: 1.0131

LeNet Test Accuracy: 65.37%

Training AlexNet...

Epoch 1/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 35s 35ms/step - accuracy: 0.2023 - loss: 2.0769 - val_accuracy: 0.4743 - val_loss: 1.4014

Epoch 2/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 22ms/step - accuracy: 0.5016 - loss: 1.3683 - val_accuracy: 0.5890 - val_loss: 1.1505

Epoch 3/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 23ms/step - accuracy: 0.6025 - loss: 1.1049 - val_accuracy: 0.6674 - val_loss: 0.9436

Epoch 4/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 22ms/step - accuracy: 0.6761 - loss: 0.9304 - val_accuracy: 0.6813 - val_loss: 0.8969

Epoch 5/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 22ms/step - accuracy: 0.7077 - loss: 0.8356 - val_accuracy: 0.6990 - val_loss: 0.8794

Epoch 6/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 22ms/step - accuracy: 0.7446 - loss: 0.7299 - val_accuracy: 0.7231 - val_loss: 0.7958

Epoch 7/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 21ms/step - accuracy: 0.7719 - loss: 0.6584 - val_accuracy: 0.7215 - val_loss: 0.8047

Epoch 8/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 21ms/step - accuracy: 0.7931 - loss: 0.6028 - val_accuracy: 0.7295 - val_loss: 0.7962

Epoch 9/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 17s 22ms/step - accuracy: 0.8111 - loss: 0.5511 - val_accuracy: 0.7225 - val_loss: 0.8584

Epoch 10/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 21ms/step - accuracy: 0.8282 - loss: 0.4939 - val_accuracy: 0.7369 - val_loss: 0.8006

AlexNet Test Accuracy: 73.69%

Training VGG...

Epoch 1/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 25s 23ms/step - accuracy: 0.2746 - loss: 1.9119 - val_accuracy: 0.5638 - val_loss: 1.1842

Epoch 2/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - accuracy: 0.5714 - loss: 1.2022 - val_accuracy: 0.6663 - val_loss: 0.9599

Epoch 3/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 21s 16ms/step - accuracy: 0.6731 - loss: 0.9333 - val_accuracy: 0.7105 - val_loss: 0.8389

Epoch 4/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 13s 16ms/step - accuracy: 0.7320 - loss: 0.7790 - val_accuracy: 0.7432 - val_loss: 0.7475

Epoch 5/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 13s 16ms/step - accuracy: 0.7729 - loss: 0.6614 - val_accuracy: 0.7560 - val_loss: 0.7139

Epoch 6/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 15ms/step - accuracy: 0.8003 - loss: 0.5712 - val_accuracy: 0.7694 - val_loss: 0.6812

Epoch 7/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - accuracy: 0.8278 - loss: 0.5027 - val_accuracy: 0.7720 - val_loss: 0.6865

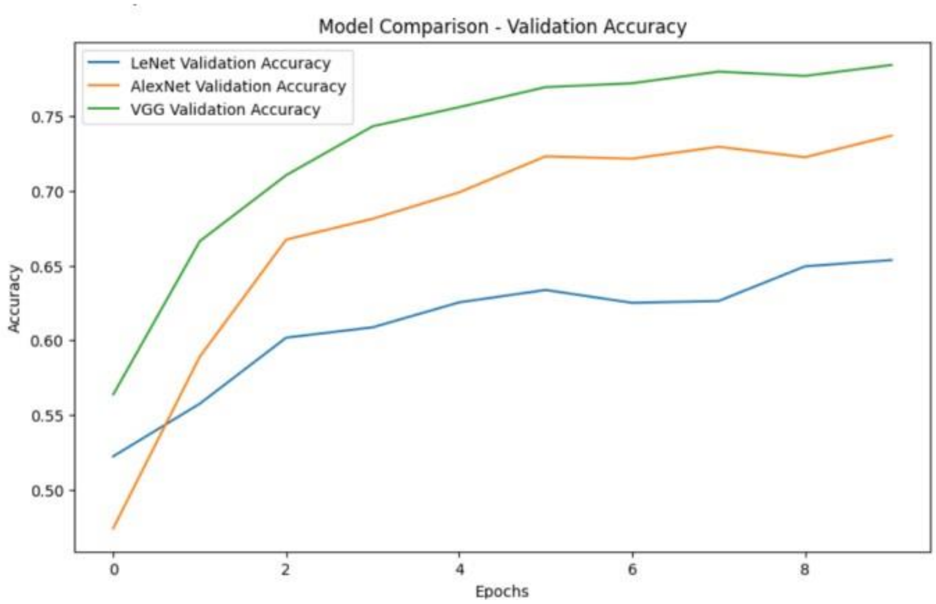
Epoch 8/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 15ms/step - accuracy: 0.8459 - loss: 0.4394 - val_accuracy: 0.7798 - val_loss: 0.6947

Epoch 9/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - accuracy: 0.8653 - loss: 0.3932 - val_accuracy: 0.7769 - val_loss: 0.7354

Epoch 10/10 782/782 ━━━━━━━━━━━━━━━━━━━━ 13s 16ms/step - accuracy: 0.8769 - loss: 0.3557 - val_accuracy: 0.7842 - val_loss: 0.7063

VGG Test Accuracy: 78.42%

Summary of Results:



LeNet Accuracy: 65.37%

AlexNet Accuracy: 73.69%

VGG Accuracy: 78.42%

The best performing model is VGG with an accuracy of 78.42%

CONCLUSION:

We successfully implemented and compared LeNet, AlexNet, and VGG-like architectures for classifying CIFAR-10 images. The performance of the models was analyzed, and the best architecture was identified.

Experiment - 10

Aim: Design and implement a simple RNN model with Keras / TensorFlow and check accuracy.

Context:

Text prediction is one of the most crucial applications of NLP.It finds its use in Image Caption Generation, Chatbots, Email applications , etc where the text is predicted based on the preview statements.

Problem Statement:

A language based model is trained on the text of Alice in Wonderland to predict the next characters given 10 previous characters.

DataSet:

Title: Alice's adventure in wonderland

Author: Lewis Caroll

Language: English

Character set encoding : ASCII

The dataset consists of textual data which has about 7 chapters and is 148.57KB.

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from tensorflow.keras.optimizers import RMSprop
import requests
import warnings
warnings.filterwarnings("ignore")

# Step 1: Load and Preprocess the Dataset
# Download the dataset
url = "https://www.gutenberg.org/files/11/11-0.txt"
response = requests.get(url)
text = response.text

# Preprocessing: Remove non-ASCII characters
```

```python
text = ''.join([char for char in text if ord(char) < 128])

# Define constants
SEQLEN = 10  # Sequence length (number of previous characters to consider)
STEP = 1     # Step size for creating sequences

# Create character-to-index and index-to-character mappings
chars = sorted(list(set(text)))
nb_chars = len(chars)
char_to_idx = {char: idx for idx, char in enumerate(chars)}
idx_to_char = {idx: char for idx, char in enumerate(chars)}

# Create input and label sequences
inputs = []
labels = []
for i in range(0, len(text) - SEQLEN, STEP):
   inputs.append(text[i:i + SEQLEN])
   labels.append(text[i + SEQLEN])

# Vectorize inputs and labels
X = np.zeros((len(inputs), SEQLEN, nb_chars), dtype=np.bool_)
y = np.zeros((len(inputs), nb_chars), dtype=np.bool_)

for i, seq in enumerate(inputs):
   for t, char in enumerate(seq):
      X[i, t, char_to_idx[char]] = 1
   y[i, char_to_idx[labels[i]]] = 1

# Step 2: Define and Compile the RNN Model
# Define the RNN model
model = Sequential([
   SimpleRNN(128, input_shape=(SEQLEN, nb_chars), unroll=True),
   Dense(nb_chars, activation='softmax')
])

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer=RMSprop(learning_rate=0.01))
model.summary()
# Step 3: Train the Model and Test Performance
# Train the model for 100 epochs, testing output every 4 epochs
for iteration in range(1,26):
   print(f"\nIteration {iteration}")
   model.fit(X, y, batch_size=128, epochs=1)
   # Generate text to test the model
```

```python
start_idx = np.random.randint(0, len(text) - SEQLEN - 1)
test_sequence = text[start_idx: start_idx + SEQLEN]
print(f"Seed text: {test_sequence}")
# Predict the next 100 characters
for _ in range(25):
    x_pred = np.zeros((1, SEQLEN, nb_chars))
    for t, char in enumerate(test_sequence):
        x_pred[0, t, char_to_idx[char]] = 1
    pred = model.predict(x_pred, verbose=0)
    next_char = idx_to_char[np.argmax(pred)]
    print(next_char, end='')
    test_sequence = test_sequence[1:] + next_char
print("\n")
```

Output:

Model: "sequential_5"

Layer (type)	Output Shape	Param #
simple_rnn_5 (SimpleRNN)	(None, 128)	25,600
dense_5 (Dense)	(None, 71)	9,159

Total params: 34,759 (135.78 KB)

Trainable params: 34,759 (135.78 KB)

Non-trainable params: 0 (0.00 B)

Iteration 1
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - loss: 3.3470
Seed text: et in the
the the the the the the t
Iteration 2
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.4148
Seed text: a feather
 there the the the the th

Iteration 3
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.2474
Seed text: the very
and and and and and and a
Iteration 4
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.2293
Seed text: the trial
the Dit it the daid the d
Iteration 5
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.2104
Seed text: yawning a
nd the peen, seen, the Ce
Iteration 6
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.2116
Seed text: hing yet,
said the Mor Alid the the
Iteration 7
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.2316
Seed text: the dance
 suen and and and and and
Iteration 8
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.2062
Seed text: uite_ as m
add and hadd hall cand th
Iteration 9
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1823
Seed text: all the r
oun hat here hath mut her
Iteration 10
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1953
Seed text:
here dir
t and and and and and and
Iteration 11
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1689
Seed text: bits. I al
l thith thith thith thith
Iteration 12
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1864
Seed text: and looked
 harking and the Duch, an
Iteration 13
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1771

Seed text: Turtle, c
e Me ind in ind in ind in
Iteration 14
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.2621
Seed text: n a long,
eeveed leed leeded leed l
Iteration 15
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.2541
Seed text: many out-o
f it bitht bid thing thit
Iteration 16
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.2101
Seed text: what it m
orl laid the wall litt la
Iteration 17
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1889
Seed text: at
the fl
e the did the did the did
Iteration 18
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1920
Seed text: race-cour
sesss she she she she she
Iteration 19
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1864
Seed text: is of fin
g and and and and and and
Iteration 20
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1572
Seed text: had been
learelle were were were w
Iteration 21
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1734
Seed text: ally good
wisting with the with the
Iteration 22
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1551
Seed text: d low-spir
e abaid Alice be abad a b
Iteration 23
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1684
Seed text: his, she w
ith the the the the the t

Iteration 24
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1458
Seed text: ure a serp
ing tid it it it it it it
Iteration 25
1132/1132 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 2.1848
Seed text: sister; W
he wore.
Wore.
Werp

Platform used: Kaggle

Scope for improvement

The model can be further extended with generating next 3-5 characters for the given input text. We can also rather use pre defined already tested models which bear a high accuracy to build the same.

Conclusion

A simple RNN model has been designed and implemented using tensorflow or keras

Experiment – 11

Aim: Design and implement an LSTM model with TensorFlow /Keras and check accuracy.

Context: Sentiment Analysis is the most common text classification tool that analyses an incoming message and tells whether the underlying sentiment is positive, negative or neutral.

Problem Statement:
To design an LSTM model to perform sentiment analysis on the given dataset.

Dataset:
This is a dataset of 25,000 movie reviews from IMDB, labelled by sentiment (positive/negative). Reviews have been preprocessed, and each review is encoded as a list of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset.

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Load the IMDB dataset
num_words = 2000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)
# Preprocessing: Pad sequences
max_review_length = 250
X_train = pad_sequences(X_train, maxlen=max_review_length)
X_test = pad_sequences(X_test, maxlen=max_review_length)
# Define the model
embedding_vector_length = 32
model = Sequential()
model.add(Embedding(num_words, embedding_vector_length,
input_length=max_review_length))
model.add(Dropout(0.2))
model.add(LSTM(32))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
```

```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
# Evaluate the model
accuracy = model.evaluate(X_test, y_test, verbose=2)[1]
print(f"Test Accuracy: {accuracy:.4f}")
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 80s 99ms/step - accuracy: 0.7087 - l
oss: 0.5298 - val_accuracy: 0.8577 - val_loss: 0.3446
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 79s 101ms/step - accuracy: 0.8684 - l
oss: 0.3139 - val_accuracy: 0.8670 - val_loss: 0.3093
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 81s 104ms/step - accuracy: 0.8853 - l
oss: 0.2749 - val_accuracy: 0.8526 - val_loss: 0.3620
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 79s 102ms/step - accuracy: 0.8995 - l
oss: 0.2478 - val_accuracy: 0.8776 - val_loss: 0.3067
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 78s 100ms/step - accuracy: 0.9072 - l
oss: 0.2279 - val_accuracy: 0.8789 - val_loss: 0.3498
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 78s 100ms/step - accuracy: 0.9066 - l
oss: 0.2279 - val_accuracy: 0.8802 - val_loss: 0.3208
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 79s 100ms/step - accuracy: 0.9225 - l
oss: 0.1990 - val_accuracy: 0.8787 - val_loss: 0.2980
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 78s 100ms/step - accuracy: 0.9293 - l
oss: 0.1815 - val_accuracy: 0.8804 - val_loss: 0.3308
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 78s 100ms/step - accuracy: 0.9313 - l
oss: 0.1787 - val_accuracy: 0.8767 - val_loss: 0.3339
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 79s 101ms/step - accuracy: 0.9315 - l
oss: 0.1737 - val_accuracy: 0.8777 - val_loss: 0.3654
782/782 - 18s - 23ms/step - accuracy: 0.8777 - loss: 0.3654
Test Accuracy: 0.8532%
```

Platform Used: Kaggle

Scope for Improvement:

We can check if our model is overfit/underfit/perfect fit and add hypertuning parameters accordingly. If the LSTM model is not yielding good accuracy then we can go for GRU model.

Conclusion:

An LSTM model has been constructed to classify the given text as a positive,negative or neutral statement.

Experiment – 12

Aim: Design and implement a GRU model with TensorFlow / Keras and check accuracy.

Context: Sentiment Analysis is the most common text classification tool that analyses an incoming message and tells whether the underlying sentiment is positive, negative or neutral.

Problem Statement:
To design a GRU model to perform sentiment analysis on the given dataset.

Dataset:
This is a dataset of 25,000 movie reviews from IMDB, labelled by sentiment (positive/negative). Reviews have been preprocessed, and each review is encoded as a list of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset.

```python
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GRU, Dense, Dropout
# Load the dataset
num_words = 5000  # Only consider the top 5000 words
maxlen = 250  # Maximum length of a review
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=num_words)
# Preprocess the data by padding sequences
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
# Define the GRU model
embedding_vector_length = 32
model = Sequential([
   Embedding(input_dim=num_words, output_dim=embedding_vector_length,
input_length=maxlen),
   Dropout(0.2),
   GRU(32),
   Dense(256, activation='relu'),
   Dropout(0.2),
   Dense(1, activation='sigmoid')
])
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```python
# Train the model
history = model.fit(
    x_train, y_train,
    validation_data=(x_test, y_test),
    epochs=10,  # You can increase epochs for better results
    batch_size=32
)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

Output:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/10

782/782 ━━━━━━━━━━━━━━━━━━━━ 104s 129ms/step - accuracy: 0.6724 - loss: 0.5537 - val_accuracy: 0.8440 - val_loss: 0.3545
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 100s 128ms/step - accuracy: 0.8830 - loss: 0.2857 - val_accuracy: 0.8781 - val_loss: 0.3054
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 141s 127ms/step - accuracy: 0.9216 - loss: 0.2059 - val_accuracy: 0.8656 - val_loss: 0.3075
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 102s 130ms/step - accuracy: 0.9350 - loss: 0.1680 - val_accuracy: 0.8811 - val_loss: 0.2920
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 104s 133ms/step - accuracy: 0.9494 - loss: 0.1320 - val_accuracy: 0.8732 - val_loss: 0.3279
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 106s 135ms/step - accuracy: 0.9590 - loss: 0.1053 - val_accuracy: 0.8748 - val_loss: 0.3599
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 102s 131ms/step - accuracy: 0.9688 - loss: 0.0878 - val_accuracy: 0.8702 - val_loss: 0.4276
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 101s 129ms/step - accuracy: 0.9744 - loss: 0.0704 - val_accuracy: 0.8726 - val_loss: 0.4214
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 142s 129ms/step - accuracy: 0.9772 -
```

```
loss: 0.0640 - val_accuracy: 0.8707 - val_loss: 0.5109
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 100s 128ms/step - accuracy: 0.9820 -
loss: 0.0516 - val_accuracy: 0.8722 - val_loss: 0.5125
782/782 ━━━━━━━━━━━━━━━━━━━━ 20s 26ms/step - accuracy: 0.8684 -
loss: 0.5352
Test Accuracy: 0.8722 %
```

Platform Used: Kaggle

Scope for Improvement:

We can check if our model is overfit/underfit/perfect fit and add hypertuning parameters accordingly.

Conclusion:

A GRU model has been constructed to classify the given text as a positive,negative or neutral statement.