1.)TF VERSION

```
import tensorflow as tf
print(tf.__version__)
import tensorflow as tf
devices = tf.config.list_physical_devices()
print("Number of devices available: ", len(devices))
for device in devices:
    print(device)
    if "GPU" in device.device_type:
        print("GPU is available")
    else:
        print("GPU is not available")
```

2.)b)RANdom NUMber generator

```
import tensorflow as tf
tf.random.set_seed(42)
tensor1 = tf.random.normal(shape=(2, 3))
tensor2 = tf.random.normal(shape=(2, 3))
print("Are the two tensors identical? ", tf.reduce_all(tensor1 == tens
or2))
```

c)

```
import tensorflow as tf
tf.random.set_seed(42)
tensor1 = tf.random.normal(shape=(2, 3))
tf.random.set_seed(11)
tensor2 = tf.random.normal(shape=(2, 3))
print("Are the two tensors identical? ", tf.reduce_all(tensor1 == tens
or2))
```

3.)Shuffling of tensors

```
#Without seed:
import tensorflow as tf
input_tensor = tf.constant([[1, 2], [3, 4], [5, 6], [7, 8]])
shuffled_tensor = tf.random.shuffle(input_tensor)
print("Shuffled tensor without seed:\n", shuffled_tensor.numpy())
#with seed
import tensorflow as tf
input_tensor = tf.constant([[1, 2], [3, 4], [5, 6], [7, 8]])
tf.random.set_seed(42)
shuffled_tensor = tf.random.shuffle(input_tensor)
print("Shuffled tensor with seed:\n", shuffled_tensor.numpy())
import tensorflow as tf
```

```
input_tensor = tf.constant([[1, 2], [3, 4], [5, 6], [7, 8]])
tf.random.set_seed(42)
shuffled_tensor_op_seed = tf.random.shuffle(input_tensor, seed=1
1)
tf.random.set_seed(42)
shuffled_tensor_global_seed = tf.random.shuffle(input_tensor)
tf.random.set_seed(42)
shuffled_tensor_global_seed2 = tf.random.shuffle(input_tensor)
print("Shuffled tensor with operation seed:\n", shuffled_tensor_op_
seed.numpy())
print("Shuffled tensor with global seed:\n", shuffled_tensor_global_
seed.numpy())
print("Shuffled tensor with global seed (second execution):\n", shuf
fled_tensor_global_seed2.numpy())
```

4.)RESHAPING TENSORS

```
import numpy as np
import tensorflow as tf
vector = np.arange(1, 25)
tensor_rank3 = tf.reshape(vector, shape=(2, 3, 4))
print("Reshaped Tensor (Rank 3):\n", tensor_rank3.numpy())
import tensorflow as tf
tensor_rank2 = tf.constant([[1, 2], [3, 4]])
tensor_rank3_newaxis = tensor_rank2[:, :, tf.newaxis]
print("Tensor (Rank 2):\n", tensor_rank2.numpy())
print("Tensor (Rank 3) with newaxis:\n", tensor_rank3_newaxis.nu
mpy())
import tensorflow as tf
tensor_rank2 = tf.constant([[1, 2], [3, 4]])
tensor_rank3_expand_dims = tf.expand_dims(tensor_rank2, axis=-
1)
print("Tensor (Rank 2):\n", tensor_rank2.numpy())
print("Tensor (Rank 3) with expand_dims:\n", tensor_rank3_expand
_dims.numpy())
```

5.)LINEAR REGREESION

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense
from sklearn.metrics import mean_squared_error, r2_score
df = pd.read_csv('/content/drive/MyDrive/Housing_data.csv')
X = df.drop('price', axis=1)
```

```
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu')
)
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, y_train, epochs=10, batch_size=10)
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print('Mean squared error:', mse)
print('R2 score:', r2)
```

6.)REGULARISATION

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
data = fetch_california_housing()
X_train, X_test, y_train, y_test = train_test_split(data.data, data.targ
et, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
ridge = Ridge(alpha=1.0)
ridge.fit(X_train_scaled, y_train)
ridge_train_predictions = ridge.predict(X_train_scaled)
ridge_test_predictions = ridge.predict(X_test_scaled)
ridge_train_mse = mean_squared_error(y_train, ridge_train_predict
ions)
ridge_test_mse = mean_squared_error(y_test, ridge_test_predictio
ns)
print("Ridge regression:")
print(f"Train MSE: {ridge_train_mse:.2f}")
print(f"Test MSE: {ridge_test_mse:.2f}")
lasso = Lasso(alpha=1.0)
lasso.fit(X_train_scaled, y_train)
lasso_train_predictions = lasso.predict(X_train_scaled)
lasso_test_predictions = lasso.predict(X_test_scaled)
```

```
lasso_train_mse = mean_squared_error(y_train, lasso_train_predictions)
lasso_test_mse = mean_squared_error(y_test, lasso_test_predictions)
print("Lasso regression:")
print(f"Train MSE: {lasso_train_mse:.2f}")
print(f"Test MSE: {lasso_test_mse:.2f}")
```

7.)ANN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
    Dense(4, activation='tanh', input_shape=(2,)),
    Dense(2, activation='tanh'),
    Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
inputs = Input(shape=(2,))
x = Dense(4, activation='tanh')(inputs)
x = Dense(2, activation='tanh')(x)
outputs = Dense(1, activation='sigmoid')(x)
model = Model(inputs=inputs, outputs=outputs)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

8.)CNN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D,
Flatten, Dense
from tensorflow.keras.models import Model
inputs = Input(shape=(28,28,1))
x = Conv2D(32, (3,3), activation='relu')(inputs)
x = MaxPooling2D((2,2))(x)
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
outputs = Dense(10, activation='softmax')(x)
model = Model(inputs=inputs, outputs=outputs)
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.summary()
import tensorflow as tf
(train_images, train_labels), (test_images, test_labels) = tf.keras.dat
asets.cifar100.load_data(label_mode='fine')
print('Number of images in the training set:', train_images.shape[0])
print('Number of images in the test set:', test_images.shape[0])
print('Resolution of the images:', train_images.shape[1:3])
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0
train_images = tf.expand_dims(train_images, axis=-1)
test_images = tf.expand_dims(test_images, axis=-1)
print(train_images.shape)
print(test_images.shape)
```

9.)pos tags

```
import spacy
nlp = spacy.load('en_core_web_sm')
text = "The quick brown fox jumps over the lazy dog."
doc = nlp(text)
for token in doc:
    print(token.text, token.pos_)
```

10.)common tags

```
import spacy
nlp = spacy.load('en_core_web_sm')
text = "The quick brown fox jumps over the lazy dog."
doc = nlp(text)
tags = [token.tag_ for token in doc]
tag_counts = {}
for tag in tags:
    if tag in tag_counts:
        tag_counts[tag] += 1
```

```
    else:
        tag_counts[tag] = 1
sorted_tags = sorted(tag_counts.items(), key=lambda x: x[1], revers
e=True)
print("Top 5 most common tags:")
for tag, count in sorted_tags[:5]:
    print(f"{tag}: {count}")
```

11.) NER

```
import spacy
nlp = spacy.load("en_core_web_sm")
text = "Apple is looking at buying U.K. startup for $1 billion"
doc = nlp(text)
for ent in doc.ents:
    print(ent.text, ent.label_)
```

12.)sigmoid

```
import numpy as np
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import LSTM, Embedding, Dense
from keras.utils import pad_sequences
max_features = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=ma
x_features)
maxlen = 500
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)
model = Sequential()
model.add(Embedding(max_features, 128, input_length=maxlen))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=32, epochs=1, validation_data
=(X_test, y_test))
score, acc = model.evaluate(X_test, y_test, batch_size=32)
print('Test score:', score)
print('Test accuracy:', acc)
```

13.)adam

```
import numpy as np
from keras.datasets import imdb
from keras.utils import pad_sequences
from keras.models import Sequential
```

```
from keras.layers import Dense, Embedding, LSTM
from keras.optimizers import Adam
max_features = 5000
maxlen = 100
batch_size = 64
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=ma
x_features)
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)
model = Sequential()
model.add(Embedding(max_features, 128, input_length=maxlen))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer=Adam(lr=1e-
4), metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=batch_size, epochs=1, validat
ion_data=(X_test, y_test))
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

14.)RMSProp

```
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding
from keras.optimizers import RMSprop
from keras.utils import pad_sequences
from keras.datasets import imdb
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=100
00)\max_len = 100
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
model = Sequential()
model.add(Embedding(10000, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=RMSprop(lr=0.001), loss='binary_crossen
tropy', metrics=['acc'])
history = model.fit(x_train, y_train, epochs=1, batch_size=128, valid
ation_split=0.2)
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```