8.

```python
import numpy as np

def one_hot_encode_words(text):
    words = text.lower().split()
    unique_words = sorted(list(set(words)))
    word_to_int = {word: i for i, word in enumerate(unique_words)}
    vocab_size = len(unique_words)

    one_hot_encoded_vectors = {}
    all_vectors = []
    for word in words:
        vector = np.zeros(vocab_size, dtype=int)
        vector[word_to_int[word]] = 1
        one_hot_encoded_vectors[word] = vector
        all_vectors.append(vector)
    return one_hot_encoded_vectors, word_to_int, np.array(all_vectors)

print("--- Word One-Hot Encoding Example ---")
sentence = "The cat sat on the mat"
word_vectors, word_vocab, word_array = one_hot_encode_words(sentence)

print("Original Sentence:", sentence)
for word, vector in word_vectors.items():
    print(f"'{word}': {vector}")

print("\nEncoded sequence of vectors for the sentence:")
print(word_array)
print("-" * 40 + "\n")
```

```
--- Word One-Hot Encoding Example ---
Original Sentence: The cat sat on the mat
 'the': [0 0 0 0 1]
 'cat': [1 0 0 0 0]
 'sat': [0 0 0 1 0]
'on': [0 0 1 0 0]
'mat': [0 1 0 0 0]
 Encoded sequence of vectors for the sentence:
 [[0 0 0 0 1] [1 0 0 0 0] [0 0 0 1 0] [0 0 1 0 0] [0 0 0 0 1] [0 1 0 0 0]]
```

2.

```
import tensorflow as tf
from tensorflow import keras
from keras.datasets import imdb
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, GlobalAveragePooling1D, Dense, Dropout

VOCAB_SIZE = 10000
MAX_LEN = 256
EMBEDDING_DIM = 16
BATCH_SIZE = 512
EPOCHS = 10

print("Loading IMDB dataset...")
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=VOCAB_SIZE)

print(f"Number of training sequences: {len(train_data)}")
print(f"Number of testing sequences: {len(test_data)}")

print("Padding sequences...")
train_data = pad_sequences(train_data, maxlen=MAX_LEN, padding='post', truncating='post')
test_data = pad_sequences(test_data, maxlen=MAX_LEN, padding='post', truncating='post')

print(f"Shape of training data after padding: {train_data.shape}")
print(f"Shape of testing data after padding: {test_data.shape}")

print("Building the model...")
model = Sequential([
    Embedding(input_dim=VOCAB_SIZE, output_dim=EMBEDDING_DIM,
input_shape=[MAX_LEN]),
    GlobalAveragePooling1D(),
    Dense(16, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
print("Compiling the model...")
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()

print("\n--- Training the model ---")
x_val = train_data[:10000]
partial_x_train = train_data[10000:]
y_val = train_labels[:10000]
```

```
partial_y_train = train_labels[10000:]

history = model.fit(
   partial_x_train,
   partial_y_train,
   epochs=EPOCHS,
   batch_size=BATCH_SIZE,
   validation_data=(x_val, y_val),
   verbose=1
)

print("\n--- Evaluating the model ---")
results = model.evaluate(test_data, test_labels, verbose=2)
print(f"\nTest Loss: {results[0]:.4f}")
print(f"Test Accuracy: {results[1]:.4f}")

prediction = model.predict(test_data[0:1])
print(f"\nPrediction for first test review: {prediction[0][0]:.4f}")
print(f"Actual label for first test review: {test_labels[0]}")
print("A prediction > 0.5 is considered positive, and <= 0.5 is negative.")
```

```
(25000, 256)
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ embedding (Embedding)           │ (None, 256, 16)        │       160,000 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ global_average_pooling1d        │ (None, 16)             │             0 │
│ (GlobalAveragePooling1D)        │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 16)             │           272 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 16)             │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 1)              │            17 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
Total params: 160,289 (626.13 KB) Trainable params: 160,289 (626.13 KB) Non-trainable params: 0 (0.00 B)
--- Evaluating the model --- 782/782 - 2s - 2ms/step - accuracy: 0.8389 - loss: 0.4250 Test Loss: 0.4250 Test Accuracy: 0.8389 1/1
━━━━━━━━━━━━━━━━━━━━ 0s 215ms/step Prediction for first test review: 0.411
```

4.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]

feature_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
                 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT']
X = pd.DataFrame(data, columns=feature_names)
y = pd.Series(target, name='MEDV')

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=[X_train.shape[1]]),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
])

model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=['mean_absolute_error'])

model.summary()

history = model.fit(
    X_train_scaled, y_train,
    epochs=100,
    validation_split=0.2,
    verbose=1
)
```

```
loss, mae = model.evaluate(X_test_scaled, y_test, verbose=0)

print("\n--- Model Evaluation ---")
print(f"Mean Absolute Error on Test Data: {mae:.2f}")

test_predictions = model.predict(X_test_scaled).flatten()

print("\n--- Example Predictions ---")
for i in range(5):
    print(f"Predicted Price: {test_predictions[i]:.2f}, Actual Price: {y_test.iloc[i]:.2f}")
```

```
Model: "sequential_2"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape               ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ dense_4 (Dense)                      │ (None, 64)                 │             896 │
├──────────────────────────────────────┼────────────────────────────┼─────────────────┤
│ dense_5 (Dense)                      │ (None, 64)                 │           4,160 │
├──────────────────────────────────────┼────────────────────────────┼─────────────────┤
│ dense_6 (Dense)                      │ (None, 1)                  │              65 │
└──────────────────────────────────────┴────────────────────────────┴─────────────────┘
 Total params: 5,121 (20.00 KB)
 Trainable params: 5,121 (20.00 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/10011/11 ━━━━━━━━━━━━━━━━━━━━ 3s 130ms/step - loss: 602.7761
- mean_absolute_error: 22.5738 - val_loss: 525.9553 - val_mean_absolute_error: 21.3836

Epoch 100/100 11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 7.1968 -
mean_absolute_error: 1.9888 - val_loss: 13.6237 - val_mean_absolute_error: 2.6591 --- Model
Evaluation --- Mean Absolute Error on Test Data: 2.32 4/4 ━━━━━━━━━━━━━━━━━━━━ 0s
62ms/step --- Example Predictions --- Predicted Price: 27.37, Actual Price: 23.60 Predicted Price:
33.55, Actual Price: 32.40 Predicted Price: 16.75, Actual Price: 13.60 Predicted Price: 26.21, Actual
Price: 22.80 Predicted Price: 16.09, Actual Price: 16.10
```

6.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import os
import shutil
import matplotlib.pyplot as plt
from PIL import Image

base_dir = 'cats_and_dogs_small'
if os.path.exists(base_dir):
    shutil.rmtree(base_dir)
os.makedirs(base_dir)

train_dir = os.path.join(base_dir, 'train')
os.makedirs(train_dir)
validation_dir = os.path.join(base_dir, 'validation')
os.makedirs(validation_dir)

train_cats_dir = os.path.join(train_dir, 'cats')
os.makedirs(train_cats_dir)
train_dogs_dir = os.path.join(train_dir, 'dogs')
os.makedirs(train_dogs_dir)
validation_cats_dir = os.path.join(validation_dir, 'cats')
os.makedirs(validation_cats_dir)
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
os.makedirs(validation_dogs_dir)

def create_dummy_files(directory, prefix, num_files):
    for i in range(num_files):
        with open(os.path.join(directory, f'{prefix}.{i}.jpg'), 'w') as f:
            f.write(f'Dummy: {prefix} {i}')

create_dummy_files(train_cats_dir, 'cat', 100)
create_dummy_files(train_dogs_dir, 'dog', 100)
create_dummy_files(validation_cats_dir, 'cat', 50)
create_dummy_files(validation_dogs_dir, 'dog', 50)

def create_pixel_image(directory):
    for filename in os.listdir(directory):
        if filename.endswith(".jpg"):
            img = Image.new('RGB', (150, 150), color='black')
            img.save(os.path.join(directory, filename))

create_pixel_image(train_cats_dir)
create_pixel_image(train_dogs_dir)
create_pixel_image(validation_cats_dir)
create_pixel_image(validation_dogs_dir)

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
validation_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)
validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dropout(0.5),
    Dense(512, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

```
model.summary()

model.compile(
    loss='binary_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
    metrics=['accuracy']
)

history = model.fit(
    train_generator,
    steps_per_epoch=5,
    epochs=15,
    validation_data=validation_generator,
    validation_steps=2,
    verbose=2
)

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(acc))

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and Validation Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'ro', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation Loss')
plt.legend()

plt.suptitle('CNN Training Metrics')
plt.show()

shutil.rmtree(base_dir)
util.rmtree(base_dir)
```

10.

```
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense

max_features = 10000
maxlen = 500
batch_size = 32

print("Loading data...")
(input_train, y_train), (input_test, y_test) = imdb.load_data(num_words=max_features)
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')

print("Pad sequences (samples x time)")
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)

print("\nBuilding the RNN model...")
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.build(input_shape=(None, maxlen))
model.summary()

print("\nCompiling the model...")
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

print("\nTraining the model...")
history = model.fit(input_train, y_train, epochs=10, batch_size=batch_size,
validation_split=0.2)

print("\nEvaluating the model on the test set...")
loss, accuracy = model.evaluate(input_test, y_test, batch_size=batch_size)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```

1.

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

input_size = 784
hidden_size1 = 512
hidden_size2 = 256
num_classes = 10
num_epochs = 15
batch_size = 128
learning_rate = 0.001

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = torchvision.datasets.MNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = torchvision.datasets.MNIST(root='./data', train=False, transform=transform)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)

def imshow(img):
    img = img / 2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()

dataiter = iter(train_loader)
images, labels = next(dataiter)
print("Visualizing a batch of training data:")
imshow(torchvision.utils.make_grid(images[:8]))
```

```
print('Labels: ', ' '.join(f'{labels[j]}' for j in range(8)))

class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.dropout1 = nn.Dropout(0.2)
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()
        self.dropout2 = nn.Dropout(0.2)
        self.fc3 = nn.Linear(hidden_size2, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu1(out)
        out = self.dropout1(out)
        out = self.fc2(out)
        out = self.relu2(out)
        out = self.dropout2(out)
        out = self.fc3(out)
        return out

model = NeuralNet(input_size, hidden_size1, hidden_size2, num_classes).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

print("\nStarting training...")
n_total_steps = len(train_loader)
for epoch in range(num_epochs):
    model.train()
    for i, (images, labels) in enumerate(train_loader):
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if (i+1) % 200 == 0:
            print(f'Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{n_total_steps}], Loss: 
{loss.item():.4f}')
```

```
print("Finished training.")

print("\nEvaluating model on test data...")
with torch.no_grad():
    model.eval()
    n_correct = 0
    n_samples = 0
    for images, labels in test_loader:
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        n_samples += labels.size(0)
        n_correct += (predicted == labels).sum().item()
    acc = 100.0 * n_correct / n_samples
    print(f'Accuracy of the network on the 10,000 test images: {acc:.2f} %')
```

3.

```
import numpy as np
from tensorflow.keras.datasets import reuters
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

print("Loading Reuters dataset...")
(train_data, train_labels), (test_data, test_labels) = reuters.load_data(num_words=10000)
print(f"Number of training samples: {len(train_data)}")
print(f"Number of testing samples: {len(test_data)}")
print(f"Number of categories: {len(np.unique(train_labels))}")

print("--- Sample training data (integer sequence) ---")
print(train_data[0])

word_index = reuters.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_newswire = ' '.join([reverse_word_index.get(i - 3, '?') for i in train_data[0]])
print("\n--- Decoded sample newswire ---")
print(decoded_newswire)

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

print("\nVectorizing data...")
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)

print("Building the neural network model...")
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(10000,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(46, activation='softmax'))

print("Compiling the model...")
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
x_val = x_train[:1000]
partial_x_train = x_train[1000:]
y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]
print("\nTraining the model...")
history = model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=512,
validation_data=(x_val, y_val), verbose=1)
print("\nEvaluating the model on the test set...")
results = model.evaluate(x_test, one_hot_test_labels, batch_size=512)
print("--- Test Results ---")
print(f"Test Loss: {results[0]:.4f}")
print(f"Test Accuracy: {results[1]:.4f}")
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']
epochs = range(1, len(loss_values) + 1)
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, acc_values, 'bo', label='Training acc')
plt.plot(epochs, val_acc_values, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.tight_layout()
plt.show()
print("\nMaking predictions on the test set...")
predictions = model.predict(x_test)
predicted_class = np.argmax(predictions[0])
print(f"\nSample Prediction for the first test newswire:")
print(f"Predicted class index: {predicted_class}")
print(f"Actual class index: {test_labels[0]}")
print(f"Confidence: {predictions[0][predicted_class]:.2%}")
```

9.

```
import tensorflow as tf
from tensorflow import keras
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, GlobalAveragePooling1D, Dense
import numpy as np

vocab_size = 10000
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=vocab_size)

word_index = imdb.get_word_index()
word_index = {k: (v + 3) for k, v in word_index.items()}
word_index["<PAD>"] = 0
word_index["<START>"] = 1
word_index["<UNK>"] = 2
word_index["<UNUSED>"] = 3
reverse_word_index = {value: key for key, value in word_index.items()}

def decode_review(text):
   return ' '.join([reverse_word_index.get(i, '?') for i in text])

print("--- Example Decoded Review ---")
print(decode_review(train_data[0]))
print("Label:", train_labels[0])
print("-" * 30)

max_length = 256
train_data = pad_sequences(train_data, value=word_index["<PAD>"], padding='post',
maxlen=max_length)
test_data = pad_sequences(test_data, value=word_index["<PAD>"], padding='post',
maxlen=max_length)

embedding_dim = 16
model = Sequential([
   Embedding(vocab_size, embedding_dim),
   GlobalAveragePooling1D(),
   Dense(16, activation='relu'),
   Dense(1, activation='sigmoid')
])
model.build(input_shape=(None, max_length))
model.summary()
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(
    train_data,
    train_labels,
    epochs=30,
    batch_size=512,
    validation_split=0.2,
    verbose=1
)

results = model.evaluate(test_data, test_labels, verbose=2)
print("\n--- Model Evaluation ---")
print(f"Test Loss: {results[0]}")
print(f"Test Accuracy: {results[1]}")
print("-" * 30)

def preprocess_text(text):
    words = text.lower().split()
    encoded_review = [word_index.get(word, 2) for word in words]
    padded_review = pad_sequences([encoded_review], value=word_index["<PAD>"],
padding='post', maxlen=max_length)
    return padded_review

positive_review = "this movie was fantastic I really enjoyed it and would recommend it to
everyone"
preprocessed_positive = preprocess_text(positive_review)
prediction_positive = model.predict(preprocessed_positive)
print(f"Prediction for positive review: {prediction_positive[0][0]}")

negative_review = "it was a terrible movie I would not recommend it to anyone"
preprocessed_negative = preprocess_text(negative_review)
prediction_negative = model.predict(preprocessed_negative)
print(f"Prediction for negative review: {prediction_negative[0][0]}")
```

7.

```
import numpy as np
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input,
decode_predictions
from tensorflow.keras.preprocessing import image
from PIL import Image
import requests
from io import BytesIO
import matplotlib.pyplot as plt

print("Loading VGG16 model...")
model = VGG16(weights='imagenet')
print("Model loaded successfully.")

image_url =
'https://upload.wikimedia.org/wikipedia/commons/4/45/A_small_cup_of_coffee.JPG'

print(f"Loading image from URL: {image_url}")
headers = {'User-Agent': 'Mozilla/5.0'}
response = requests.get(image_url, headers=headers)
response.raise_for_status()
img = Image.open(BytesIO(response.content))
img_resized = img.resize((224, 224))
img_array = image.img_to_array(img_resized)
img_batch = np.expand_dims(img_array, axis=0)
img_preprocessed = preprocess_input(img_batch)

print("Classifying the image...")
predictions = model.predict(img_preprocessed)
decoded_predictions = decode_predictions(predictions, top=3)[0]
print("Classification complete.")

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(img)
plt.title("Original Image")
plt.axis('off')

plt.subplot(1, 2, 2)
y_pos = np.arange(len(decoded_predictions))
performance = [pred[2] for pred in decoded_predictions]
class_names = [pred[1] for pred in decoded_predictions]
plt.barh(y_pos, performance, align='center', color='skyblue')
plt.yticks(y_pos, class_names)
plt.gca().invert_yaxis()
plt.xlabel('Probability')
plt.title('Top 3 Predictions')
for index, value in enumerate(performance):
    plt.text(value, index, f"{value:.2f}")
plt.tight_layout()
plt.show()

print("\n--- Top 3 Predictions ---")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i+1}: {label} ({score:.2%})")
```

5.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
print("x_train shape:", x_train.shape, "y_train shape:", y_train.shape, "x_test shape:",
x_test.shape, "y_test shape:", y_test.shape)

np.random.seed(0)

plt.xticks([])
plt.yticks([])
plt.xlabel([y_train[1]])
plt.imshow(x_train[1], cmap=plt.cm.binary)
plt.show()

x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(64, (2, 2), strides=(1, 1), padding='same', activation='relu',
input_shape=(28, 28, 1)))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Conv2D(32, (2, 2), strides=(1, 1), padding='same', activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Dropout(0.3))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
model.summary()

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

model_log = model.fit(x_train, y_train, batch_size=60, epochs=10, verbose=1,
validation_split=0.3)

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

```
predictions = model.predict(x_test)

x_test = x_test.reshape(-1, 28, 28)

plt.xticks([])
plt.yticks([])
plt.xlabel([y_train[0]])
plt.imshow(x_test[0], cmap=plt.cm.binary)
plt.show()

plt.figure()
plt.subplot(2, 1, 1)
plt.plot(model_log.history['accuracy'])
plt.plot(model_log.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='lower right')

plt.subplot(2, 1, 2)
plt.plot(model_log.history['loss'])
plt.plot(model_log.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.tight_layout()
plt.show()

def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(img, cmap=plt.cm.binary)
    predicted_label = np.argmax(predictions_array)
    color = 'blue' if predicted_label == true_label else 'red'
    plt.xlabel(f"{[predicted_label]} {100*np.max(predictions_array):.0f}% ({[true_label]})",
color=color)

def plot_value_array(i, predictions_array, true_label):
    predictions_array, true_label = predictions_array[i], true_label[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    bars = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)
    bars[predicted_label].set_color('red')
    bars[true_label].set_color('blue')

num_rows, num_cols = 5, 3
num_images = num_rows * num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))

for i in range(num_images):
    plt.subplot(num_rows, 2*num_cols, 2*i + 1)
    plot_image(i, predictions, y_test, x_test)
    plt.subplot(num_rows, 2*num_cols, 2*i + 2)
    plot_value_array(i, predictions, y_test)

plt.show()
```