Boston

```
# Import libraries for data manipulation, machine learning, and visualization
import pandas as pd                    # For data handling using DataFrames
import numpy as np                     # For numerical operations
import tensorflow as tf                # For building and training neural networks# Import required libraries
import pandas as pd                            # Pandas for data manipulation and loading CSVs
import numpy as np                             # NumPy for numerical operations
import tensorflow as tf                        # TensorFlow for building and training models
from sklearn.model_selection import train_test_split # Used to split data into training and testing sets
import matplotlib.pyplot as plt                # Used to visualize training progress
import warnings                                # Used to manage warning messages
warnings.filterwarnings('ignore')              # Ignore warnings in output
# Load the IMDB dataset from CSV
data = pd.read_csv('imdb_dataset.csv')         # Load dataset from CSV file
print(f"Original shape: {data.shape}")         # Print initial shape of the dataset
# Preprocessing (Optional)
print(data.isnull().sum())                     # Count and print null values per column
print(data.duplicated().sum())                 # Print total number of duplicate rows
# Drop rows with missing values in 'review' or 'sentiment', and remove duplicates based on 'review'
data = data.dropna(subset=['review', 'sentiment']).drop_duplicates(subset=['review'])
print(f"Shape after removing Null & duplicates: {data.shape}")  # Print shape after cleaning
# Encode labels ('positive' -> 1, 'negative' -> 0)
data['sentiment'] = data['sentiment'].map({'positive': 1, 'negative': 0})  # Map string labels to integers
data['sentiment'] = (data['sentiment'] == 'positive').astype(int)          # Ensures only 1s and 0s (redundant but safe)
print(data['sentiment'].value_counts())                # Print number of positive and negative reviews
# Separate features and target
X = data['review'].values                      # Feature: reviews (text)
y = data['sentiment'].values                   # Target: sentiment (0 or 1)
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)       # random_state ensures reproducibility
print(f"Training set shape: {X_train.shape}Test set shape: {X_test.shape}")  # Print split shapes
# Text vectorization
max_words = 10000                              # Max number of unique words to consider
max_len = 200                                  # Max sequence length (padding/truncation)
vectorizer = tf.keras.layers.TextVectorization(        # Create text vectorization layer
    max_tokens=max_words,                      # Limit vocabulary size
    output_mode='int',                         # Convert text to integers
    output_sequence_length=max_len             # Fix sequence length to 200 tokens
)
vectorizer.adapt(X_train)                      # Learn the vocabulary from training text
# Build the neural network model
model = tf.keras.Sequential([                  # Create a sequential model
    vectorizer,                                # First layer: text vectorization
    tf.keras.layers.Embedding(max_words, 16, input_length=max_len),  # Word embedding (dense vector of size 16)
    tf.keras.layers.GlobalAveragePooling1D(),           # Pooling layer that averages over word embeddings
    tf.keras.layers.Dense(16, activation='relu'),       # Hidden dense layer with ReLU activation
    tf.keras.layers.Dense(1, activation='sigmoid')      # Output layer with sigmoid for binary classification
])
# Compile the model
model.compile(
    optimizer='adam',                  # Adam optimizer for efficient gradient descent
    loss='binary_crossentropy',        # Binary classification loss function
    metrics=['accuracy']               # Track accuracy during training and evaluation
)
# Train the model
history = model.fit(
    X_train, y_train,                  # Input features and labels
    epochs=10,                         # Train for 10 full passes over the training data
    batch_size=32,                     # Train in mini-batches of 32 samples
    validation_split=0.2,              # Use 20% of training data for validation
    verbose=1                          # Display training progress
)
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)  # Evaluate on test set (no output)
print(f'\nTest Loss: {test_loss:.4f}')                  # Print test loss (rounded to 4 decimal places)
print(f'Test Accuracy: {test_acc:.2%}')                 # Print test accuracy as percentage
# Plot results
plt.figure(figsize=(12, 4))                    # Set figure size for plots
# Plot training and validation loss
plt.subplot(1, 2, 1)                           # First subplot
plt.plot(history.history['loss'], label='Training Loss')        # Plot training loss over epochs
plt.plot(history.history['val_loss'], label='Validation Loss')  # Plot validation loss over epochs
plt.title('Model Loss')                        # Title for loss graph
plt.xlabel('Epoch')                            # X-axis label
plt.ylabel('Loss')                             # Y-axis label
plt.legend()                                   # Add legend to distinguish lines
# Plot training and validation accuracy
plt.subplot(1, 2, 2)                           # Second subplot
plt.plot(history.history['accuracy'], label='Training Accuracy')       # Training accuracy
plt.plot(history.history['val_accuracy'], label='Validation Accuracy') # Validation accuracy
plt.title('Model Accuracy')                    # Title for accuracy graph
plt.xlabel('Epoch')                            # X-axis label
plt.ylabel('Accuracy')                         # Y-axis label
plt.legend()                                   # Add legend
```

IMDB

```
# Import libraries for data manipulation, machine learning, and visualization
from sklearn.model_selection import train_test_split  # To split dataset into training and testing sets
from sklearn.preprocessing import StandardScaler      # To normalize features before training
import matplotlib.pyplot as plt            # For plotting training metrics and predictions
import warnings                            # To manage warning messages
warnings.filterwarnings('ignore')          # Ignore any warnings to keep output clean
# Load the Boston Housing dataset from CSV
data = pd.read_csv('Boston.csv')           # Read dataset into a pandas DataFrame
data.shape                                 # Display the number of rows and columns in the dataset
# Preprocessing (Optional)
print(data.isnull().sum())                 # Print number of missing values in each column
data = data.dropna()                       # Remove rows with any missing values

print(data.duplicated().sum())             # Print number of duplicate rows
data = data.drop_duplicates()              # Remove duplicate rows from the dataset

print(f"Shape after removing Null & duplicates: {data.shape}")  # Print new shape of cleaned dataset

print(data.columns)                        # Display column names of the dataset
# Assuming the target column is named 'medv' (adjust if different)
X = data.drop('medv', axis=1).values       # Features: drop the target column 'medv'
y = data['medv'].values                    # Target: the column 'medv' as a NumPy array
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y,                                  # Features and target
    test_size=0.2,                         # 20% of data for testing
    random_state=42                        # Seed to ensure reproducible split
)
# Scale the features
scaler = StandardScaler()                  # Create a StandardScaler instance for normalization
X_train_scaled = scaler.fit_transform(X_train)  # Fit to training data and scale
X_test_scaled = scaler.transform(X_test)        # Scale test data using same parameters
# Build the neural network model
model = tf.keras.Sequential([                   # Sequential model (layers added in order)
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),  # First hidden layer with 64 neurons
    tf.keras.layers.Dense(32, activation='relu'),  # Second hidden layer with 32 neurons
    tf.keras.layers.Dense(1)                       # Output layer for regression (1 neuron, linear activation)
])
# Compile the model
model.compile(
    optimizer='adam',       # Optimizer: Adam (adaptive gradient descent)
    loss='mse',             # Loss function: Mean Squared Error (good for regression)
    metrics=['mae']         # Metric to track: Mean Absolute Error (easier to interpret)
)
# Train the model
history = model.fit(
    X_train_scaled, y_train,    # Training features and labels
    epochs=100,                 # Train for 100 iterations through the dataset
    batch_size=32,              # Use mini-batches of 32 samples
    validation_split=0.2,       # 20% of training data used for validation
    verbose=1                   # Show progress bar during training
)
# Evaluate the model
test_loss, test_mae = model.evaluate(X_test_scaled, y_test, verbose=0)  # Evaluate on unseen test data
print(f"\nTest Mean Absolute Error: ${test_mae:.2f}k")                  # Print the MAE formatted in thousands
# Make predictions
predictions = model.predict(X_test_scaled)  # Predict house prices using the test data
# Plot training history
plt.figure(figsize=(12, 4))  # Create a figure with 2 subplots side by side
# Plot loss (MSE) over epochs
plt.subplot(1, 2, 1)                          # First subplot
plt.plot(history.history['loss'], label='Training Loss')          # Plot training loss
plt.plot(history.history['val_loss'], label='Validation Loss')    # Plot validation loss
plt.title('Model Loss')                       # Title of plot
plt.xlabel('Epoch')                           # Label x-axis
plt.ylabel('Loss')                            # Label y-axis
plt.legend()                                  # Add legend
# Plot MAE over epochs
plt.subplot(1, 2, 2)                          # Second subplot
plt.plot(history.history['mae'], label='Training MAE')            # Plot training MAE
plt.plot(history.history['val_mae'], label='Validation MAE')      # Plot validation MAE
plt.title('Model MAE')                        # Title of plot
plt.xlabel('Epoch')                           # Label x-axis
plt.ylabel('MAE')                             # Label y-axis
plt.legend()                                  # Add legend
# Plot actual vs predicted prices
plt.figure(figsize=(8, 6))                    # New figure for actual vs predicted plot
plt.scatter(y_test, predictions, alpha=0.5)   # Scatter plot of actual vs predicted prices
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')  # Diagonal reference line
plt.xlabel('Actual Price')                    # Label x-axis
plt.ylabel('Predicted Price')                 # Label y-axis
plt.title('Actual vs Predicted Prices')       # Plot title
plt.tight_layout()                            # Automatically adjust subplot parameters
plt.show()                                    # Display the plots
```

Fashion

```python
# Import required libraries
import pandas as pd                          # Used for data manipulation and analysis
import numpy as np                            # Provides support for arrays and numerical operations
import tensorflow as tf                       # TensorFlow library for deep learning
from tensorflow import keras                  # Keras API from TensorFlow for building neural networks
import matplotlib.pyplot as plt               # Used for plotting graphs and visualizations
import warnings                               # Used to handle warning messages
warnings.filterwarnings('ignore')             # Ignore warnings in the output
# Load train and test data from CSV files
train_data = pd.read_csv('fashion-mnist_train.csv')    # Load training dataset
test_data = pd.read_csv('fashion-mnist_test.csv')      # Load testing dataset
print(f"Original train shape: {train_data.shape}, test shape: {test_data.shape}")  # Print shape of datasets
# Preprocessing (Optional)
print(train_data.isnull().sum())              # Print count of missing values per column in train dataset
print(test_data.isnull().sum())               # Print count of missing values per column in test dataset

train_data = train_data.dropna()              # Drop rows with missing values in training dataset
test_data = test_data.dropna()                # Drop rows with missing values in test dataset

print(train_data.duplicated().sum())          # Print number of duplicate rows in train dataset
print(test_data.duplicated().sum())           # Print number of duplicate rows in test dataset

train_data = train_data.drop_duplicates()     # Drop duplicate rows in training dataset
test_data = test_data.drop_duplicates()       # Drop duplicate rows in test dataset

print("shape after removing Null & duplicates")        # Print after-cleaning info
print(f"Train : {train_data.shape}")                   # Print cleaned train shape
print(f"Test : {test_data.shape}")                     # Print cleaned test shape
# Extract features and labels
x_train = train_data.drop('label', axis=1).values       # Features (images) from training data
y_train = train_data['label'].values                    # Labels (targets) from training data
x_test = test_data.drop('label', axis=1).values         # Features (images) from test data
y_test = test_data['label'].values                      # Labels (targets) from test data
# Normalize pixel values
x_train = x_train.astype('float32') / 255.0             # Convert pixel values to float and normalize (0-1)
x_test = x_test.astype('float32') / 255.0               # Normalize test set similarly
# Reshape for CNN input (28x28 images with 1 channel)
x_train = x_train.reshape(-1, 28, 28, 1)                # Reshape train data to 4D tensor for CNN
x_test = x_test.reshape(-1, 28, 28, 1)                  # Reshape test data similarly
print(f"Train shape: {x_train.shape}, Test shape: {x_test.shape}")  # Print new shape

print(train_data['label'].nunique())                    # Print number of unique classes (should be 10)
# Class names for visualization (Optional)
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',    # Human-readable class names
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle Boot']
# Visualize sample images
plt.figure(figsize=(10, 10))                            # Set figure size for plotting
for i in range(25):                                     # Plot first 25 images
    plt.subplot(5, 5, i+1)                              # Create 5x5 grid
    plt.xticks([])                                      # Remove x-axis ticks
    plt.yticks([])                                      # Remove y-axis ticks
    plt.grid(False)                                     # Remove grid lines
    plt.imshow(x_train[i], cmap=plt.cm.binary)          # Display image in grayscale
    plt.xlabel(class_names[y_train[i]])                 # Label each image with its class name
plt.savefig('sample_images.png')                        # Save the plot as an image file
# Build the CNN model
model = keras.Sequential([                              # Create a Sequential model
    keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),  # Conv layer with 32 filters, 3x3
kernel
    keras.layers.MaxPooling2D((2,2)),                   # Max pooling layer with 2x2 pool size
    keras.layers.Dropout(0.25),                         # Dropout layer for regularization (25% dropout)
    keras.layers.Conv2D(64, (3,3), activation='relu'),  # Second conv layer with 64 filters
    keras.layers.MaxPooling2D((2,2)),                   # Second pooling layer
    keras.layers.Dropout(0.25),                         # Dropout layer
    keras.layers.Conv2D(128, (3,3), activation='relu'), # Third conv layer with 128 filters
    keras.layers.Flatten(),                             # Flatten output into 1D vector for Dense layers
    keras.layers.Dense(128, activation='relu'),         # Dense layer with 128 units and ReLU activation
    keras.layers.Dropout(0.25),                         # Dropout layer
    keras.layers.Dense(10, activation='softmax')        # Output layer with 10 classes and softmax for
classification
])
# Compile the model
model.compile(optimizer='adam',                         # Adam optimizer for adaptive learning rate
              loss='sparse_categorical_crossentropy',   # Loss function for multi-class classification
              metrics=['accuracy'])                     # Metric: classification accuracy
# Train the model
history = model.fit(x_train, y_train,                   # Training data
                    epochs=10,                          # Train for 10 epochs
                    batch_size=32,                      # Use batches of 32 samples
                    validation_data=(x_test, y_test),   # Use test data for validation during training
                    verbose=1)                          # Show training progress
# Evaluate the model on test set
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)  # Get test loss and accuracy
print(f'\nTest Loss: {test_loss:.4f}')                  # Print formatted test loss
print(f'Test Accuracy: {test_acc:.2%}')                 # Print accuracy as percentage
# Plot training and validation accuracy and loss
plt.figure(figsize=(10, 5))                             # Create a new figure for plots

plt.subplot(1, 2, 1)                                    # First subplot for accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')       # Plot training accuracy
plt.plot(history.history['val_accuracy'], label='Validation Accuracy') # Plot validation accuracy
plt.xlabel('Epochs')                                    # Label x-axis
plt.ylabel('Accuracy')                                  # Label y-axis
plt.title('Training and Validation Accuracy')           # Title of the plot
plt.legend()                                            # Add legend

plt.subplot(1, 2, 2)                                    # Second subplot for loss
plt.plot(history.history['loss'], label='Training Loss')            # Plot training loss
plt.plot(history.history['val_loss'], label='Validation Loss')      # Plot validation loss
plt.xlabel('Epochs')                                    # Label x-axis
plt.ylabel('Loss')                                      # Label y-axis
plt.title('Training and Validation Loss')               # Title of the plot
plt.legend()                                            # Add legend
```