BFS-DFS:

```cpp
#include <iostream>
#include <omp.h>
#include <vector>
#include <queue>
#include <chrono>

using namespace std;
using namespace chrono;

const int MAX_VERTICES = 100;

void dfs_recursive(vector<vector<int>>& graph, int vertex, vector<bool>& visited){
    visited[vertex] = true;
    cout<<vertex<<" ";
    #pragma omp parallel for
    for (int neighbor : graph[vertex]){
        if (!visited[neighbor]){
            dfs_recursive(graph, neighbor, visited);
        }
    }
}

void dfs(vector<vector<int>>& graph, int start_vertex){
    vector<bool> visited(graph.size(), false);
    dfs_recursive(graph, start_vertex, visited);
}

void bfs(vector<vector<int>>& graph, int start_vertex){
    vector<bool> visited(graph.size(), false);
    queue<int> q;
    q.push(start_vertex);
    visited[start_vertex] = true;
    while(!q.empty()){
        int vertex = q.front();
        q.pop();
        cout<<vertex<<" ";
        #pragma omp parallel for
        for (int neighbor : graph[vertex]){
            if (!visited[neighbor]){
                #pragma omp critical
                {
                    q.push(neighbor);
                    visited[neighbor] = true;
                }
            }
        }
    }
}

int main(){
    vector<vector<int>> graph(MAX_VERTICES);
    int num_edges, num_vertices;
```

```
    cout<<"Enter number of vertices: ";
    cin>>num_vertices;
    cout<<"Enter number of edges: ";
    cin>>num_edges;

    cout<<"Enter v1 and v2:"<<endl;
    for(int i=0; i<num_edges; ++i){
        int v1, v2;
        cin>>v1>>v2;
        graph[v1].push_back(v2);
        graph[v2].push_back(v1);
    }

    int start_vertex;
    cout<<"Enter starting vertex: ";
    cin>>start_vertex;

    cout<<"\nBFS Traversal: ";
    auto start_time = high_resolution_clock::now(); // Start time measurement
    bfs(graph, start_vertex);
    auto end_time = high_resolution_clock::now(); // End time measurement
    auto duration = duration_cast<nanoseconds>(end_time - start_time); // Calculate
duration
    cout << "\nParallel BFS executed in " << duration.count() << " nanoseconds." << endl;

    cout<<"\nDFS Traversal: ";
    start_time = high_resolution_clock::now(); // Start time measurement
    dfs(graph, start_vertex);
    end_time = high_resolution_clock::now(); // End time measurement
    duration = duration_cast<nanoseconds>(end_time - start_time); // Calculate duration
    cout << "\nParallel DFS executed in " << duration.count() << " nanoseconds." << endl;
    return 0;
}
```

Bubble-Merge Sort:

```
#include<iostream>
#include<omp.h>
#include <chrono>

using namespace std;
using namespace chrono;

void bubble_sort(int arr[], int n){
    bool swapped;
    for (int i=0; i<n-1; i++){
        swapped = false;
        #pragma omp parallel for shared(arr, swapped)
        for (int j=0; j<n-i-1; j++){
            if (arr[j] > arr[j+1]){
                swap(arr[j], arr[j+1]);
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}

void merge(int arr[], int l, int m, int r){
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];

    for (int i=0; i<n1; ++i)
        L[i] = arr[l+i];
    for (int j=0; j<n2; ++j)
        R[j] = arr[m+1+j];

    int i=0, j=0, k=l;
    while(i<n1 && j<n2) {
        if (L[i] <= R[j]){
            arr[k] = L[i];
            i++;
        }
        else{
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i<n1){
        arr[k] = L[i];
        i++; k++;
    }
    while (j<n2){
        arr[k] = R[j];
        j++; k++;
    }
```

```
}

void merge_sort(int arr[], int l, int r){
    if (l<r){
        int m = l + (r-l) / 2;
        #pragma omp parallel sections
        {
            #pragma omp section
                merge_sort(arr, l, m);
            #pragma omp section
                merge_sort(arr, m+1, r);
        }
        merge(arr, l, m, r);
    }
}

int main(){
    int arr_size;
    cout<<"Enter size of the array: ";
    cin>>arr_size;

    int arr_bubble[arr_size];
    int arr_merge[arr_size];
    cout<<"Enter array elements: ";
    for(int i=0; i<arr_size; ++i){
        cin>>arr_bubble[i];
        arr_merge[i] = arr_bubble[i];
    }

    auto start_time = high_resolution_clock::now();
    bubble_sort(arr_bubble, arr_size);
    auto end_time = high_resolution_clock::now();
    auto duration1 = duration_cast<nanoseconds>(end_time - start_time);
    cout<<"The bubble sorted array: ";
    for(int i=0; i<arr_size; ++i){
        cout<<arr_bubble[i]<<" ";
    }
    cout << "\nParallel Bubble Sort executed in " << duration1.count() << " nanoseconds."
<< endl;

    start_time = high_resolution_clock::now();
    merge_sort(arr_merge, 0, arr_size-1);
    end_time = high_resolution_clock::now();
    duration1 = duration_cast<nanoseconds>(end_time - start_time);
    cout<<"The merge sorted array: ";
    for(int i=0; i<arr_size; ++i){
        cout<<arr_merge[i]<<" ";
    }
    cout << "\nParallel Merge Sort executed in " << duration1.count() << " nanoseconds."
<< endl;
    return 0;
}
```

Min-Max-Sum-Average:

```cpp
#include <iostream>
#include <omp.h>
using namespace std;

int main(){
    int n;
    cout<<"Enter number of inputs: ";
    cin>>n;
    int arr[n];
    cout<<"Enter "<<n<<" integers: ";
    for (int i=0; i<n; ++i){
        cin>>arr[i];
    }

    int sum=0, min_val=arr[0], max_val=arr[0];

    #pragma omp parallel for reduction(+:sum)
    for (int i=0; i<n; i++){
        sum += arr[i];
    }

    double avg = sum / static_cast<double>(n);

    #pragma omp parallel for reduction(min:min_val) reduction(max:max_val)
    for (int i=0; i<n; i++){
        if (min_val > arr[i]) min_val = arr[i];
        if (max_val < arr[i]) max_val = arr[i];
    }

    cout<<"\nSum: "<<sum;
    cout<<"\nAverage: "<<avg;
    cout<<"\nMin value: "<<min_val;
    cout<<"\nMax value: "<<max_val;
}
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
from keras.models import Sequential
from keras.layers import Dense

df = pd.read_csv('boston.csv')
```

```
df
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	
...	...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	
505	0.04741	0.0	11.93	0.0	0.573	6.030	NaN	2.5050	1	273	21.0	396.90	

506 rows × 14 columns

Next steps: Generate code with df View recommended plots

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

```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

```
df.fillna(df.mean(), inplace=True)
df.isnull().sum()
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

```
x = df.loc[:,df.columns!='MEDV']
y = df.loc[:,df.columns=='MEDV']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1)

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
model = Sequential([
    Dense(128, input_shape=(13,), activation='relu'),
    Dense(64, activation='relu'),
    Dense(1, activation='linear')
])

model.compile(optimizer='adam', loss='mse')
model.summary()

model.fit(x_train, y_train, epochs=100, validation_split=0.05, batch_size=4)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 128)               1792

 dense_4 (Dense)             (None, 64)                8256

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 10113 (39.50 KB)
Trainable params: 10113 (39.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/100
84/84 [==============================] - 1s 4ms/step - loss: 366.9554 - val_loss: 135.5153
Epoch 2/100
84/84 [==============================] - 0s 2ms/step - loss: 86.4951 - val_loss: 65.9189
Epoch 3/100
84/84 [==============================] - 0s 2ms/step - loss: 56.5855 - val_loss: 38.5619
Epoch 4/100
84/84 [==============================] - 0s 2ms/step - loss: 46.6949 - val_loss: 30.5471
Epoch 5/100
84/84 [==============================] - 0s 2ms/step - loss: 41.6506 - val_loss: 27.2339
Epoch 6/100
84/84 [==============================] - 0s 2ms/step - loss: 37.8727 - val_loss: 19.8974
Epoch 7/100
84/84 [==============================] - 0s 2ms/step - loss: 33.7030 - val_loss: 18.1673
Epoch 8/100
84/84 [==============================] - 0s 2ms/step - loss: 31.8399 - val_loss: 14.5055
Epoch 9/100
84/84 [==============================] - 0s 2ms/step - loss: 29.4191 - val_loss: 18.4194
Epoch 10/100
84/84 [==============================] - 0s 2ms/step - loss: 26.8320 - val_loss: 11.0481
Epoch 11/100
84/84 [==============================] - 0s 2ms/step - loss: 26.2834 - val_loss: 13.7912
Epoch 12/100
84/84 [==============================] - 0s 2ms/step - loss: 25.0446 - val_loss: 10.1749
Epoch 13/100
84/84 [==============================] - 0s 2ms/step - loss: 24.4713 - val_loss: 10.5278
Epoch 14/100
84/84 [==============================] - 0s 2ms/step - loss: 23.4797 - val_loss: 9.7537
Epoch 15/100
84/84 [==============================] - 0s 2ms/step - loss: 23.8243 - val_loss: 10.9534
Epoch 16/100
84/84 [==============================] - 0s 2ms/step - loss: 23.5469 - val_loss: 12.3875
Epoch 17/100
84/84 [==============================] - 0s 2ms/step - loss: 22.4954 - val_loss: 8.7134
Epoch 18/100
84/84 [==============================] - 0s 2ms/step - loss: 22.0781 - val_loss: 10.7995
Epoch 19/100
84/84 [==============================] - 0s 2ms/step - loss: 21.7918 - val_loss: 7.8017
Epoch 20/100
84/84 [==============================] - 0s 2ms/step - loss: 22.4719 - val_loss: 6.4616
Epoch 21/100
84/84 [==============================] - 0s 2ms/step - loss: 20.8347 - val_loss: 16.4125
Epoch 22/100
```

```
y_pred = model.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
```

```
5/5 [==============================] - 0s 3ms/step
Mean Squared Error: 9.73751543665373
Mean Absolute Error: 2.4446944970833626
```

```
y_pred = model.predict(x_test)
ps=[]
for i in y_pred:
    ps.append(list(i))

d = pd.DataFrame({'actual':y_test['MEDV'],'predicted':ps})
d
```

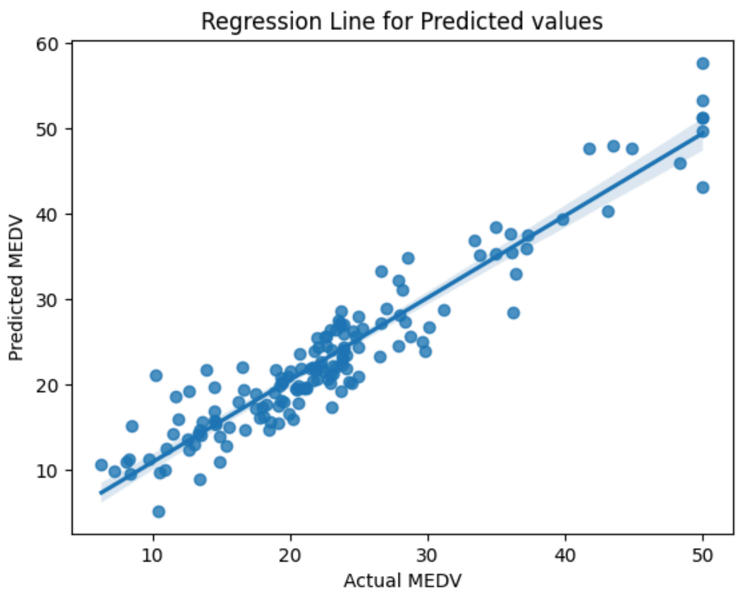
```
5/5 [==============================] - 0s 3ms/step
```

	actual	predicted
307	28.2	[31.010822]
343	23.9	[24.089771]
47	16.6	[19.299845]
67	22.0	[20.634861]
362	20.8	[21.841286]
...	...	...
467	19.1	[15.47618]
95	28.4	[27.274672]
122	20.5	[19.438705]
260	33.8	[35.16483]
23	14.5	[15.768017]

152 rows × 2 columns

Next steps: Generate code with d View recommended plots

```
sns.regplot(x=y_test, y=y_pred)
plt.title("Regression Line for Predicted values")
plt.xlabel("Actual MEDV")
plt.ylabel("Predicted MEDV")
plt.show()
```




```
Start coding or generate with AI.
```

```
import tensorflow as tf
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Flatten,MaxPooling2D, Conv2D

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()

model = Sequential([
    Conv2D(64, (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='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=5, validation_split=0.2)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
Epoch 1/5
1500/1500 [==============================] - 55s 36ms/step - loss: 1.3412 - accuracy: 0.8449 - val_loss: 0.3380 - val_accuracy: 0.87
Epoch 2/5
1500/1500 [==============================] - 52s 35ms/step - loss: 0.2890 - accuracy: 0.8959 - val_loss: 0.3272 - val_accuracy: 0.88
Epoch 3/5
1500/1500 [==============================] - 51s 34ms/step - loss: 0.2482 - accuracy: 0.9083 - val_loss: 0.3130 - val_accuracy: 0.88
Epoch 4/5
1500/1500 [==============================] - 51s 34ms/step - loss: 0.2234 - accuracy: 0.9173 - val_loss: 0.3130 - val_accuracy: 0.89
Epoch 5/5
1500/1500 [==============================] - 50s 33ms/step - loss: 0.2068 - accuracy: 0.9233 - val_loss: 0.3223 - val_accuracy: 0.89
```

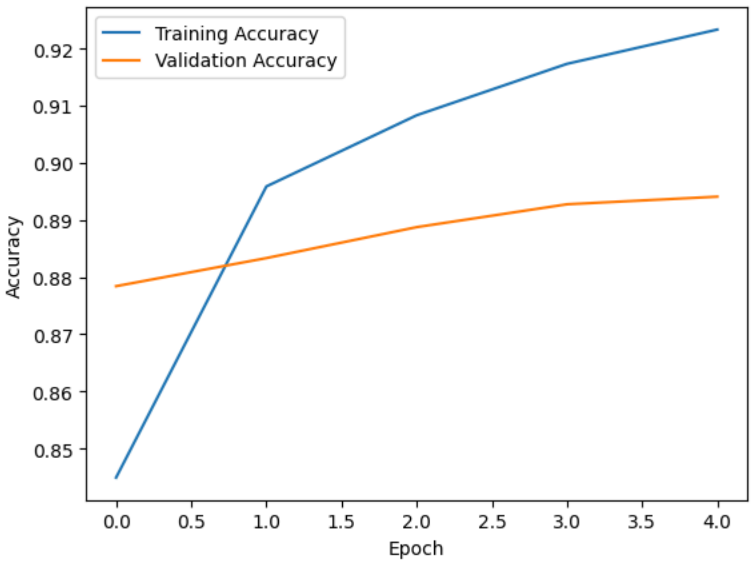
```
loss, acc = model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 2s 8ms/step - loss: 0.3583 - accuracy: 0.8860
```

```
labels = ['t_shirt', 'trouser', 'pullover', 'dress', 'coat', 'sandal', 'shirt', 'sneaker', 'bag', 'ankle_boots']
predictions = model.predict(x_test[:1])
label = labels[np.argmax(predictions)]
```

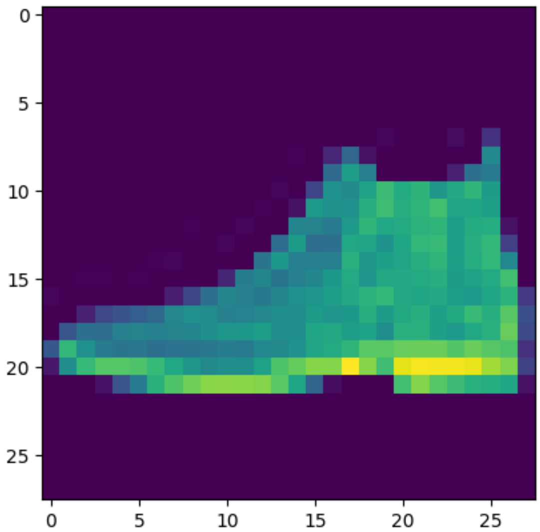
```
1/1 [==============================] - 0s 94ms/step
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```
import matplotlib.pyplot as plt
print(label)
plt.imshow(x_test[:1][0])
plt.show()
```

```
ankle_boots
```




```
Start coding or generate with AI.
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense

data = pd.read_csv('goog.csv')
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data['Close'].values.reshape(-1, 1))

def create_sequences(data, time_steps=6):
    x, y = [], []
    for i in range(len(data) - time_steps):
        x.append(data[i:i+time_steps, 0])
        y.append(data[i+time_steps, 0])
    return np.array(x), np.array(y)

x, y = create_sequences(scaled_data)

model = Sequential([
    LSTM(50, input_shape=(x.shape[1], 1)),
    Dense(1)
])
model.compile(optimizer='adam', loss='mse')
model.fit(x, y, epochs=100, batch_size=4, validation_split=0.05)
```

```
y_pred = model.predict(x)
y_pred = scaler.inverse_transform(y_pred)
y_test = scaler.inverse_transform(y.reshape(-1, 1))
```

```
2/2 [==============================] - 0s 7ms/step
```

```
last_day_price = data['Close'].values[-1]
last_6_days = data['Close'][-6:].values.reshape(-1, 1)
last_6_days_scaled = scaler.transform(last_6_days)
x_pred = last_6_days_scaled.reshape((1, 6, 1))
```

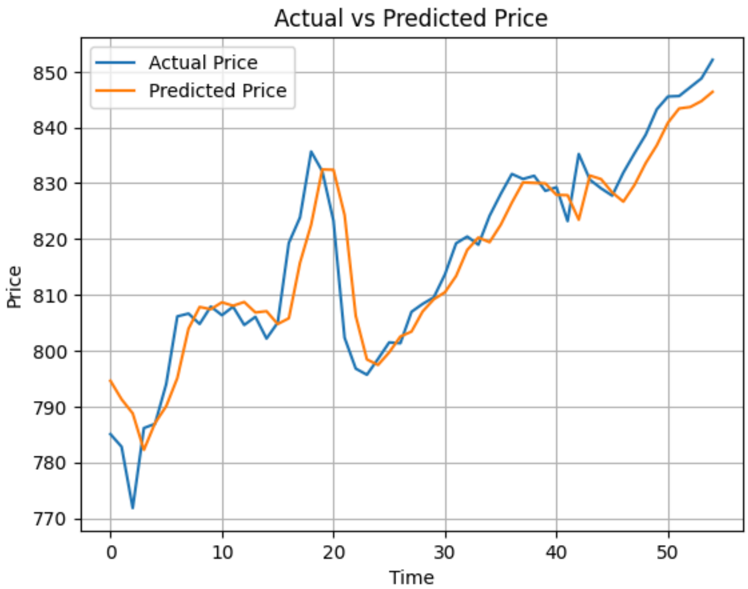
```
pred_price = model.predict(x)
pred_price = scaler.inverse_transform(pred_price)
print('Actual price for the last day:', last_day_price)
print('Predicted price for the last day:', pred_price)
```

```
2/2 [==============================] - 0s 6ms/step
Actual price for the last day: 852.119995
Predicted price for the last day: [[794.6096 ]
 [791.2908 ]
 [788.7692 ]
 [782.2278 ]
 [787.0373 ]
 [790.05524]
 [795.10034]
 [803.923  ]
 [807.8346 ]
 [807.43195]
 [808.66724]
 [808.0531 ]
 [808.72156]
 [806.85077]
 [807.0698 ]
 [804.776  ]
 [805.80817]
 [815.75275]
 [822.57227]
 [832.47284]
 [832.4112 ]
 [824.3079 ]
 [806.2377 ]
 [798.4372 ]
 [797.4483 ]
 [799.7061 ]
 [802.5358 ]
 [803.4268 ]
 [807.0362 ]
 [809.2162 ]
 [810.46857]
 [813.3809 ]
 [818.05756]
 [820.2764 ]
 [819.43646]
 [822.54675]
 [826.5223 ]
 [830.1429 ]
 [830.06006]
 [829.97235]
 [827.91736]
 [827.85406]
 [823.4629 ]
 [831.3792 ]
 [830.7409 ]
 [828.3401 ]
 [826.69965]
 [829.6906 ]
 [833.59607]
 [836.8054 ]
 [840.8485 ]
 [843.3908 ]
 [843.6539 ]
 [844.71436]
 [846.3743 ]]
```

```
# Plotting the original test data
plt.plot(y_test, label='Actual Price')

# Plotting the predicted prices
plt.plot(y_pred, label='Predicted Price')

plt.title('Actual vs Predicted Price')
plt.xlabel('Time')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```




```
Start coding or generate with AI.
```

```
import pandas as pd
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Flatten,Conv2D,MaxPooling2D
import matplotlib.pyplot as plt
```

```
train = pd.read_csv('fashion-mnist_train.csv')
test = pd.read_csv('fashion-mnist_test.csv')
x_train = train.drop(['label'],axis=1)
y_train = train['label']
x_test = test.drop(['label'],axis=1)
y_test = test['label']
x_test
```

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10
0	0	0	0	0	0	0	0	9	8	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	14	53	99	17
3	0	0	0	0	0	0	0	0	0	161
4	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
9995	0	0	0	0	0	0	0	0	0	37
9996	0	0	0	0	0	0	0	0	0	0
9997	0	0	0	0	0	0	0	0	0	0
9998	0	1	3	0	0	0	0	0	0	0
9999	0	0	0	0	0	0	0	140	119	103

10000 rows × 784 columns

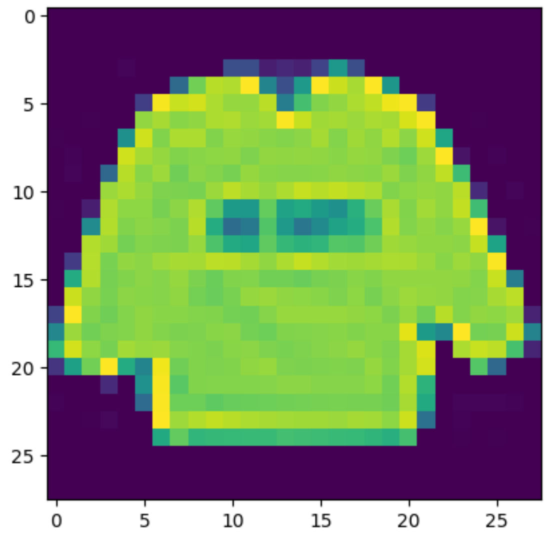
```
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape the input data to the required shape (28, 28, 1)
x_train_reshaped = x_train.values.reshape(-1, 28, 28, 1)
x_test_reshaped = x_test.values.reshape(-1, 28, 28, 1)
```

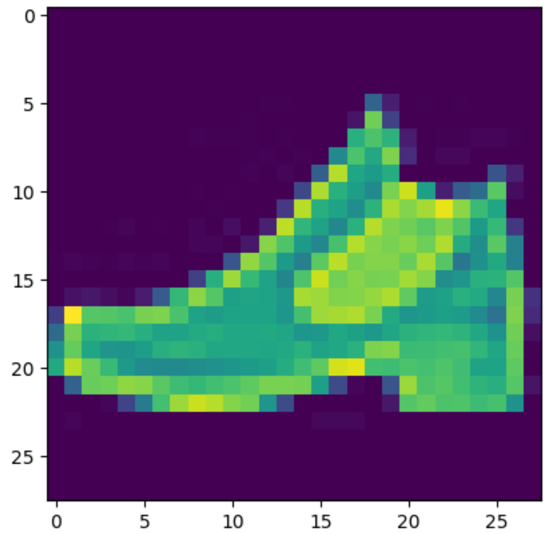
```
labels = ['t-shirt','trouser','pullover','dress','coat','sandal','sneakers',
          'shirt','bag','ankle boots']
```

```
for i in range(20):
    print(labels[y_train[i]])
    plt.imshow(x_train_reshaped[i])
    plt.show()
```

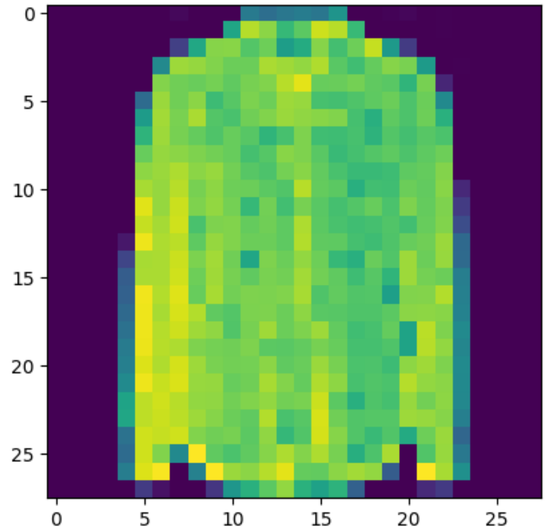
pullover



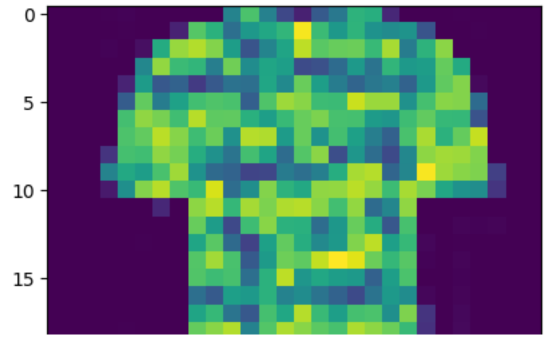
ankle boots

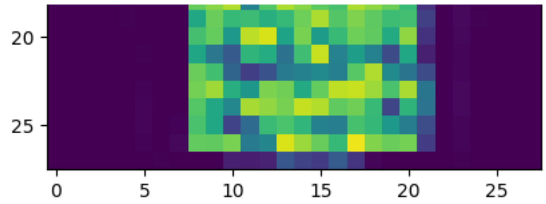


sneakers

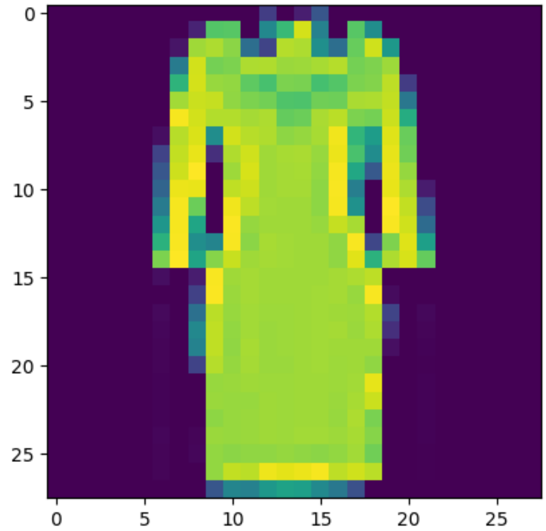


t-shirt

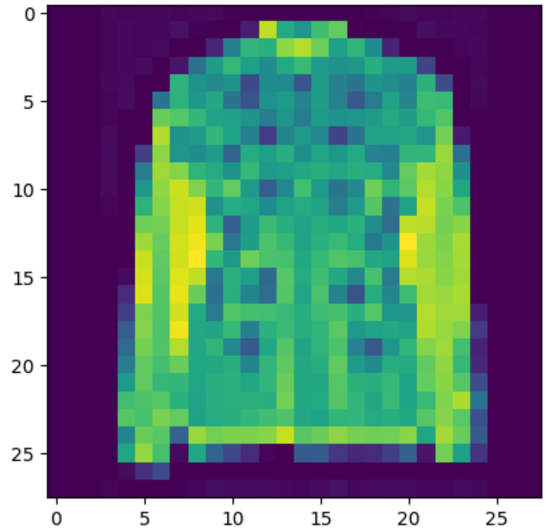




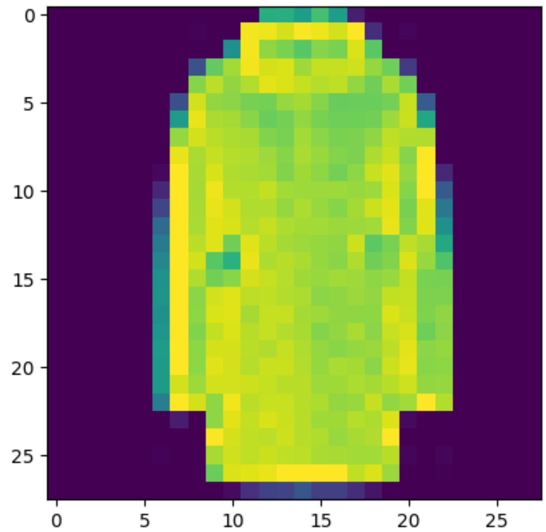
dress



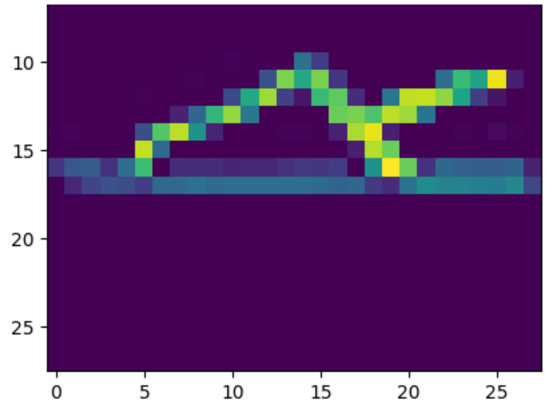
coat



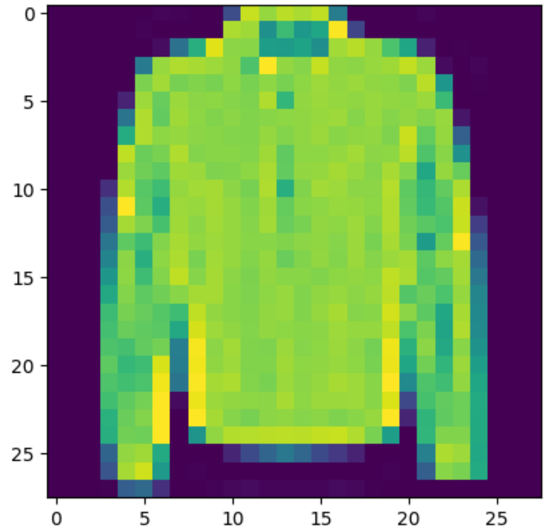
coat



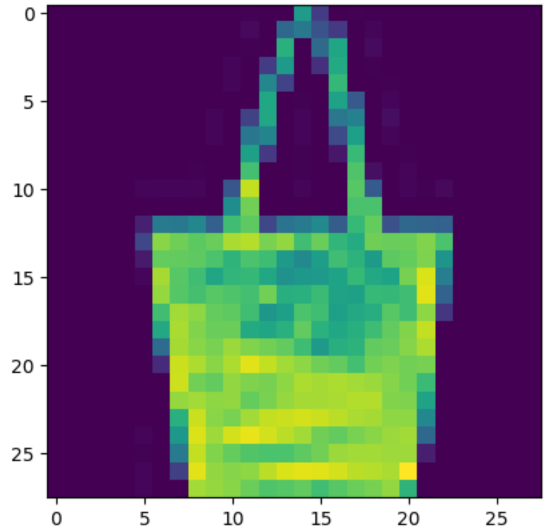
sandal



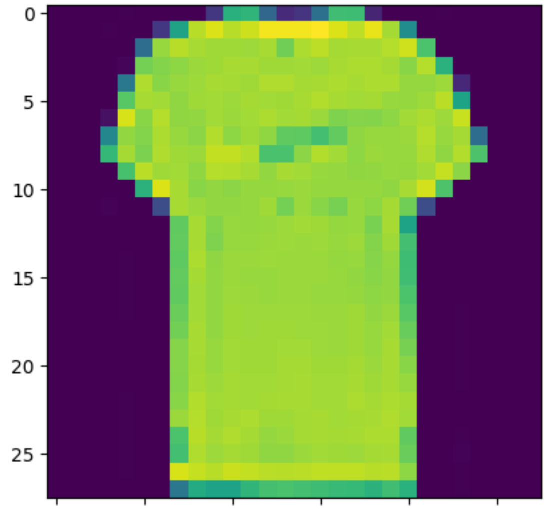
coat



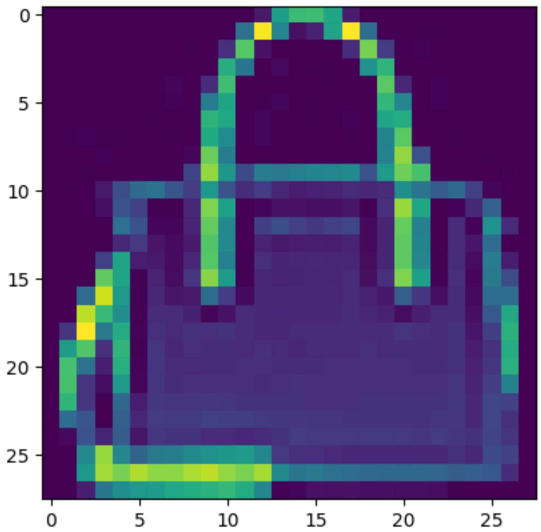
bag



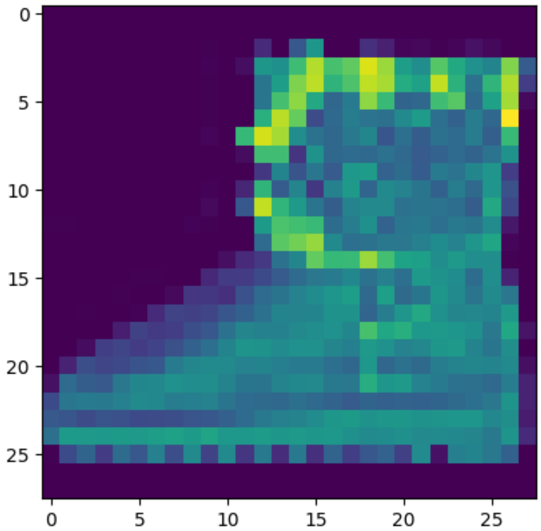
t-shirt



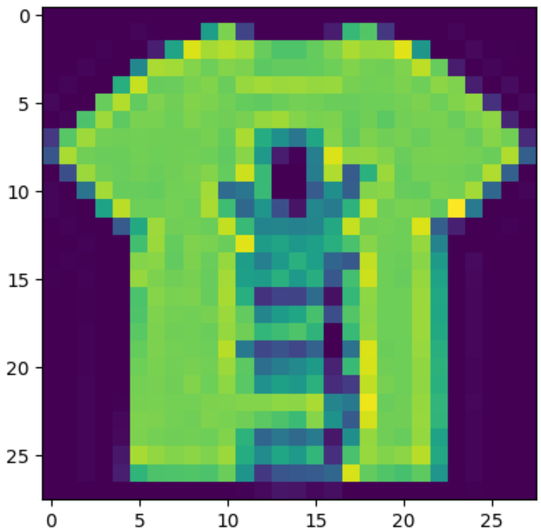
bag



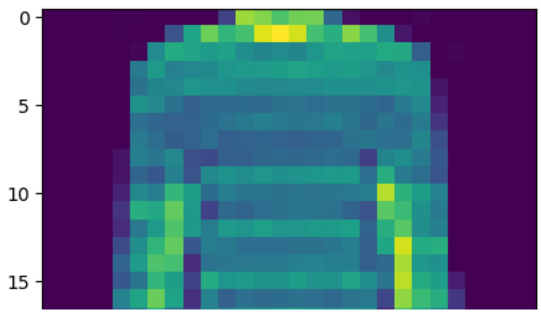
ankle boots

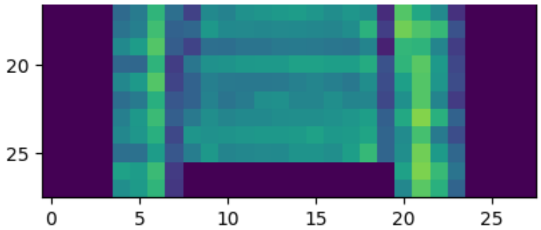


t-shirt

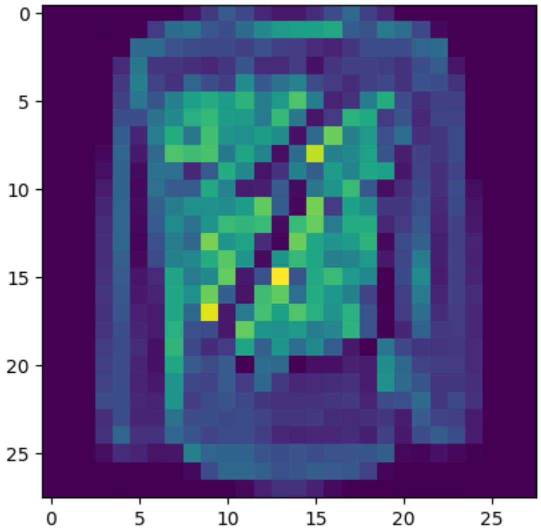


pullover

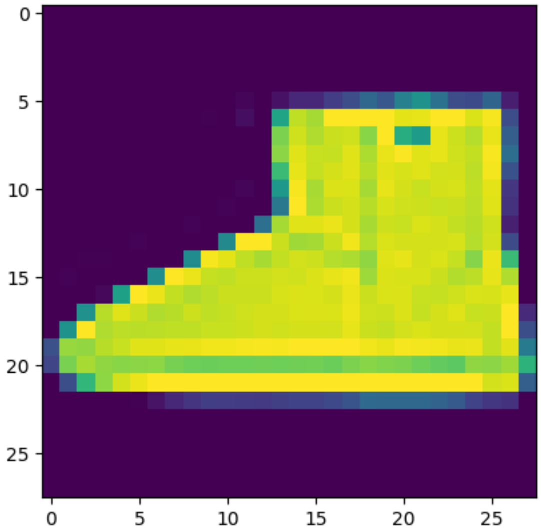




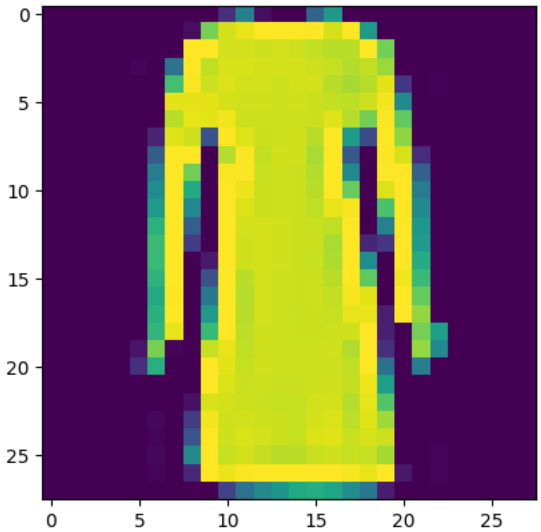
pullover



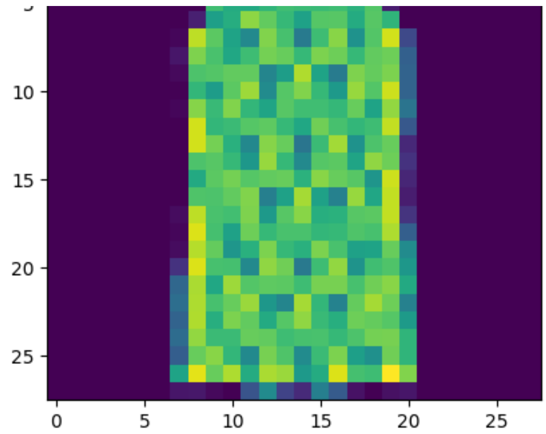
ankle boots



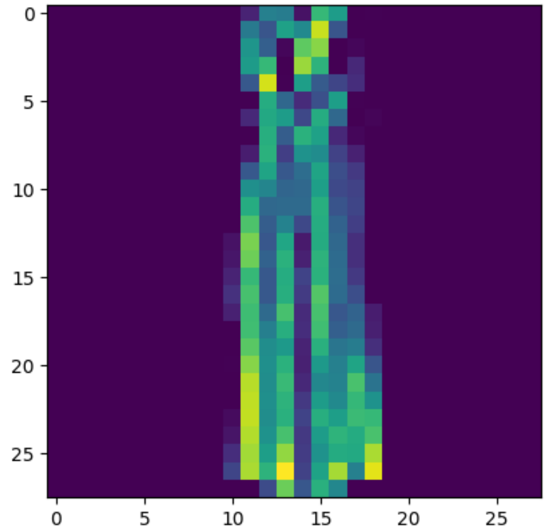
dress



dress



dress




```
model = Sequential()
```

```
model.add(Conv2D(filters=64,kernel_size=(3,3),activation='relu',input_shape=(28,28,1)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(128,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 64)        640

 max_pooling2d (MaxPooling2  (None, 13, 13, 64)        0
 D)

 flatten (Flatten)           (None, 10816)             0

 dense (Dense)               (None, 128)               1384576

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 1386506 (5.29 MB)
Trainable params: 1386506 (5.29 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.fit(x_train_reshaped, y_train, epochs=5, batch_size=32, validation_data=(x_test_reshaped, y_test))
```

```
Epoch 1/5
1875/1875 [==============================] - 77s 40ms/step - loss: 0.3809 - accuracy: 0.8643 - val_loss: 0.2788 - val_accuracy: 0.9
Epoch 2/5
1875/1875 [==============================] - 74s 40ms/step - loss: 0.2597 - accuracy: 0.9053 - val_loss: 0.2574 - val_accuracy: 0.9
Epoch 3/5
1875/1875 [==============================] - 72s 38ms/step - loss: 0.2137 - accuracy: 0.9215 - val_loss: 0.2396 - val_accuracy: 0.9
Epoch 4/5
1875/1875 [==============================] - 73s 39ms/step - loss: 0.1781 - accuracy: 0.9342 - val_loss: 0.2334 - val_accuracy: 0.9
Epoch 5/5
1875/1875 [==============================] - 72s 38ms/step - loss: 0.1486 - accuracy: 0.9455 - val_loss: 0.2353 - val_accuracy: 0.9
```

```
<keras.src.callbacks.History at 0x7bb4b9bfcfd0>
```

```
loss,acc = model.evaluate(x_test_reshaped,y_test)
```

```
313/313 [==============================] - 2s 8ms/step - loss: 0.2353 - accuracy: 0.9205
```

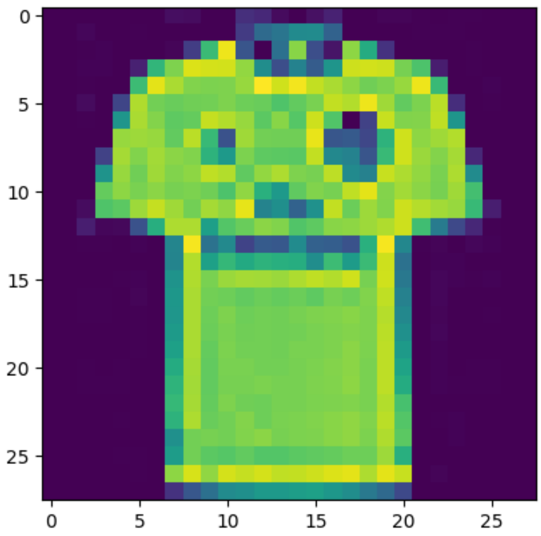
```
predictions = model.predict(x_test_reshaped[:1])
```

```
1/1 [==============================] - 0s 98ms/step
```

```
label = labels[np.argmax(predictions)]
```

```
print(label)
plt.imshow(x_test_reshaped[:1][0])
plt.show()
```

```
t-shirt
```



CUDA Matrix:

```
//execute the commands
// -> nvcc filename.cu
// -> ./a.out

//%%cu
#include <iostream>

using namespace std;

__global__ void multiply(int* A, int* B, int* C, int size) {
    // Uses thread indices and block indices to compute each element
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < size && col < size) {
        int sum = 0;
        for (int i = 0; i < size; i++) {
            sum += A[row * size + i] * B[i * size + col];
        }
        C[row * size + col] = sum;
    }
}


void initialize(int* matrix, int size) {
    for (int i = 0; i < size * size; i++) {
        matrix[i] = rand() % 10;
    }
}

void print(int* matrix, int size) {
    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            cout << matrix[row * size + col] << " ";
        }
        cout << '\n';
    }
}

int main() {
    int * A, * B, * C;

    int N = 2;
    int matrixSize = N * N;
    size_t matrixBytes = matrixSize * sizeof(int);

    A = new int[matrixSize];
    B = new int[matrixSize];
    C = new int[matrixSize];

    initialize(A, N);
    initialize(B, N);
```

```
    cout << "Matrix A: \n";
    print(A, N);

    cout << "Matrix B: \n";
    print(B, N);


    int * X, * Y, * Z;

    cudaMalloc(&X, matrixBytes);
    cudaMalloc(&Y, matrixBytes);
    cudaMalloc(&Z, matrixBytes);

    // Copy values from A to X and B to Y
    cudaMemcpy(X, A, matrixBytes, cudaMemcpyHostToDevice);
    cudaMemcpy(Y, B, matrixBytes, cudaMemcpyHostToDevice);

    // Threads per CTA dimension
    int THREADS = 2;

    // Blocks per grid dimension (assumes THREADS divides N evenly)
    int BLOCKS = N / THREADS;

    // Use dim3 structs for block  and grid dimensions
    dim3 threads(THREADS, THREADS);
    dim3 blocks(BLOCKS, BLOCKS);

    multiply<<<blocks, threads>>>(X, Y, Z, N);

    cudaMemcpy(C, Z, matrixBytes, cudaMemcpyDeviceToHost);
    cout << "Multiplication of matrix A and B: \n";
    print(C, N);

    delete[] A;
    delete[] B;
    delete[] C;

    cudaFree(X);
    cudaFree(Y);
    cudaFree(Z);

    return 0;

    // nvcc filename.cu -o filename && ./filename
}
```

CUDA Vector:

```
//execute the commands
// -> nvcc filename.cu
// -> ./a.out

//%%cu
#include <iostream>

using namespace std;

__global__ void add(int* A, int* B, int* C, int size) {
    int tid = blockIdx.x * blockDim.x + threadIdx.x;

    if (tid < size) {
        C[tid] = A[tid] + B[tid];
    }
}


void initialize(int* vector, int size) {
    for (int i = 0; i < size; i++) {
        vector[i] = rand() % 10;
    }
}

void print(int* vector, int size) {
    for (int i = 0; i < size; i++) {
        cout << vector[i] << " ";
    }
    cout << endl;
}

int main() {
    int N = 8;
    int * A, * B, * C;

    int vectorSize = N;
    size_t vectorBytes = vectorSize * sizeof(int);

    A = new int[vectorSize];
    B = new int[vectorSize];
    C = new int[vectorSize];

    initialize(A, vectorSize);
    initialize(B, vectorSize);

    cout << "Vector A: ";
    print(A, N);
    cout << "Vector B: ";
    print(B, N);

    int * X, * Y, * Z;
```

```
    cudaMalloc(&X, vectorBytes);
    cudaMalloc(&Y, vectorBytes);
    cudaMalloc(&Z, vectorBytes);

    cudaMemcpy(X, A, vectorBytes, cudaMemcpyHostToDevice);
    cudaMemcpy(Y, B, vectorBytes, cudaMemcpyHostToDevice);

    int threadsPerBlock = 256;
    int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;

    add<<<blocksPerGrid, threadsPerBlock>>>(X, Y, Z, N);

    cudaMemcpy(C, Z, vectorBytes, cudaMemcpyDeviceToHost);

    cout << "Addition: ";
    print(C, N);

    delete[] A;
    delete[] B;
    delete[] C;

    cudaFree(X);
    cudaFree(Y);
    cudaFree(Z);

    return 0;

    // nvcc filename.cu -o filename && ./filename
}
```