You need to work on a popular Fashion MNIST dataset for this HW. The dataset includes tiny images of fashion pieces. The objective is to create a set of supervised learning models that can predict the type of item based on its image.

In order to load the dataset you need to have `tensorflow V2` on your computer. Use the following code to install the package

In [ ]:
```
# !pip install --upgrade tensorflow
```

You can also check the version of it using the following code.

In [1]:
```
import tensorflow as tf
tf.__version__
```

Out[1]: '2.18.0'

Now, it's time to load the dataset

In [2]:
```
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz (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 (https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz)
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz)
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz)
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
```

As can be seen from the above code, the dataset was divided into train and test sets. Let's take a look at the X_train

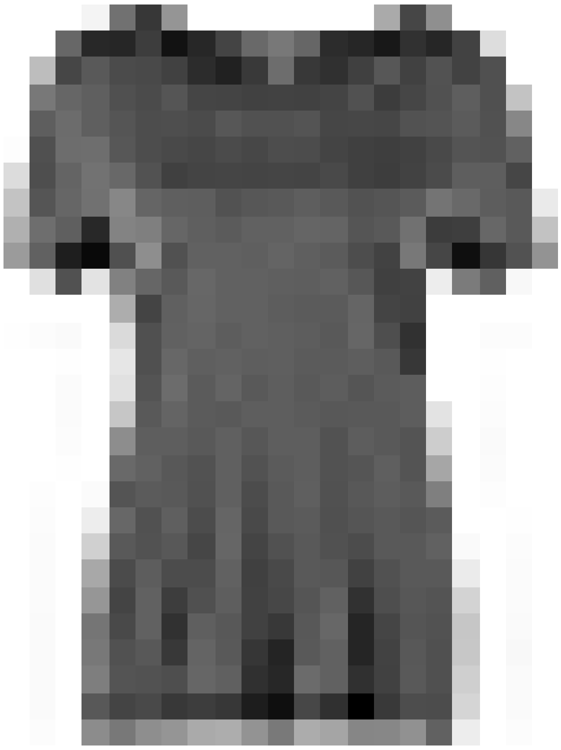
In [3]:
```
X_train.shape
```

Out[3]: (60000, 28, 28)

As it is clear, the train dataset (`X_train`) contains 60,000 images of size 28 x 28. We can visualize one of the images using the following code:

In [11]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

sample_image = X_train[10]
plt.imshow(sample_image, cmap='binary')
plt.axis('off')
plt.show()
```



The `y_train` also includes values between 0 and 9. Each represents a particular category. For example, we can check the value of `y_train` for the above image.

In [12]:
```
y_train[10]
```

Out[12]: 0

The above code shows that the image belongs to category 0. To get the associated label with each category, you can use the following code:

In [13]:
```
class_names = ['T-shirt/top','Trouser','Pullover','Dress','Coat','Sanda
print(class_names[y_train[10]])
```

```
T-shirt/top
```

Preprcessing the data

In [21]:
```
# Normalize the image data to range [0, 1]
X_train_norm = X_train / 255.0
X_test_norm = X_test / 255.0

# Flatten the 28x28 images into 784-length vectors
X_train_flat = X_train_norm.reshape(len(X_train_norm), -1)
X_test_flat = X_test_norm.reshape(len(X_test_norm), -1)

# For AUC metric (multi-class), we'll one-hot encode test labels
from sklearn.preprocessing import label_binarize
y_test_binarized = label_binarize(y_test, classes=range(10))
```

- **Task1:** Use the train set to train various supervised models and evaluate their performance using the test set.
  - Use different supervised learning models.
  - Use different metrics such as **accuracy**, **precision**, **AUC**, and ... in your model evaluation.
  - It is not enough to report the metrics. It is crucial that you interpret the metrics for each model and compare them across different models.
  - You may need to use the cross validation methods for hyperparameter selection.
  - Specify the model that outperforms the other models.
- **Task2:** Use the best model to predict your own fashion pieces.
  - Take a picture of ten fashion pieces of your own (take pictures in square format).
  - Resize images to the correct size (28,28).
  - Grayscale your images.
  - Visualize all the images side by side
  - Use the best model in Task 1 to predict the label of each of your own images.
  - How accurate is the final result?

Output

- Make sure to put descriptive comments on your code
- Use the markdown cell format in Jupiter to add your own interpretation to the result in each section.
- Make sure to keep the output of your runs when you want to save the final version of the file.
- The final work should be very well structured and should have a consistent flow of analysis.

Task 1: Train and Evaluate Multiple Supervised Models

In [15]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, accuracy_score, prec
from sklearn.model_selection import cross_val_score
```

1. Logistic Regression

In [24]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, roc_auc_sc

lr = LogisticRegression(max_iter=1000, solver='lbfgs', multi_class='mul
lr.fit(X_train_flat, y_train)
y_pred_lr = lr.predict(X_test_flat)
y_prob_lr = lr.predict_proba(X_test_flat)

print("Logistic Regression:")
print("Accuracy:", round(accuracy_score(y_test, y_pred_lr), 4))
print("Precision (macro):", round(precision_score(y_test, y_pred_lr, av
print("AUC (OvR):", round(roc_auc_score(y_test_binarized, y_prob_lr, mu
```

```
Logistic Regression:
Accuracy: 0.705
Precision (macro): 0.7005
AUC (OvR): 0.9459
```

2. K-Nearest Neighbors

In [25]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_flat, y_train)
y_pred_knn = knn.predict(X_test_flat)
y_prob_knn = knn.predict_proba(X_test_flat)

print("K-Nearest Neighbors:")
print("Accuracy:", round(accuracy_score(y_test, y_pred_knn), 4))
print("Precision (macro):", round(precision_score(y_test, y_pred_knn, a
print("AUC (OvR):", round(roc_auc_score(y_test_binarized, y_prob_knn, m
```

```
C:\Users\17kin\anaconda3\lib\site-packages\sklearn\neighbors\_classificati
on.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `
kurtosis`), the default behavior of `mode` typically preserves the axis it
acts along. In SciPy 1.11.0, this behavior will change: the default value
of `keepdims` will become False, the `axis` over which the statistic is ta
ken will be eliminated, and the value None will no longer be accepted. Set
`keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)

K-Nearest Neighbors:
Accuracy: 0.8541
Precision (macro): 0.8575
AUC (OvR): 0.9584
```

3. Support Vector Machine

In [27]:
```
from sklearn.svm import SVC

svm = SVC(probability=True)
# SVM is computationally expensive, so we use a smaller subset
svm.fit(X_train_flat[:10000], y_train[:10000])
y_pred_svm = svm.predict(X_test_flat)
y_prob_svm = svm.predict_proba(X_test_flat)

print("Support Vector Machine:")
print("Accuracy:", round(accuracy_score(y_test, y_pred_svm), 4))
print("Precision (macro):", round(precision_score(y_test, y_pred_svm, a
print("AUC (OvR):", round(roc_auc_score(y_test_binarized, y_prob_svm, m
```

```
Support Vector Machine:
Accuracy: 0.8531
Precision (macro): 0.8527
AUC (OvR): 0.9861
```

4. Random Forest

In [26]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train_flat, y_train)
y_pred_rf = rf.predict(X_test_flat)
y_prob_rf = rf.predict_proba(X_test_flat)

print("Random Forest:")
print("Accuracy:", round(accuracy_score(y_test, y_pred_rf), 4))
print("Precision (macro):", round(precision_score(y_test, y_pred_rf, av
print("AUC (OvR):", round(roc_auc_score(y_test_binarized, y_prob_rf, mu
```

```
Random Forest:
Accuracy: 0.8763
Precision (macro): 0.8756
AUC (OvR): 0.9894
```

5. MLP Neural Network

In [28]:
```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(128,), max_iter=10)
mlp.fit(X_train_flat, y_train)
y_pred_mlp = mlp.predict(X_test_flat)
y_prob_mlp = mlp.predict_proba(X_test_flat)

print(" 🔹 MLP Classifier:")
print("Accuracy:", round(accuracy_score(y_test, y_pred_mlp), 4))
print("Precision (macro):", round(precision_score(y_test, y_pred_mlp, a
print("AUC (OvR):", round(roc_auc_score(y_test_binarized, y_prob_mlp, m
```

```
🔹 MLP Classifier:
Accuracy: 0.7954
Precision (macro): 0.7902
AUC (OvR): 0.9755

C:\Users\17kin\anaconda3\lib\site-packages\sklearn\neural_network\_multila
yer_perceptron.py:692: ConvergenceWarning: Stochastic Optimizer: Maximum i
terations (10) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Summary Table

In [29]:
```
import pandas as pd

summary = {
    "Logistic Regression": [accuracy_score(y_test, y_pred_lr), precisic
    "KNN": [accuracy_score(y_test, y_pred_knn), precision_score(y_test,
    "Random Forest": [accuracy_score(y_test, y_pred_rf), precision_scor
    "SVM": [accuracy_score(y_test, y_pred_svm), precision_score(y_test,
    "MLP": [accuracy_score(y_test, y_pred_mlp), precision_score(y_test,
}

summary_df = pd.DataFrame(summary, index=["Accuracy", "Precision (Macrc
print(summary_df)
```

	Accuracy	Precision (Macro)	AUC (OvR)
Logistic Regression	0.7050	0.7005	0.9459
KNN	0.8541	0.8575	0.9584
Random Forest	0.8763	0.8756	0.9894
SVM	0.8531	0.8527	0.9861
MLP	0.7954	0.7902	0.9755

Cross Validation

In [30]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
```

1. Cross-Validation for Random Forest

In [31]:
```
param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

grid_rf = GridSearchCV(RandomForestClassifier(), param_grid_rf, cv=3, s
grid_rf.fit(X_train_flat, y_train)

print("✅ Best Random Forest Params:", grid_rf.best_params_)
print("🔎 Best Cross-Validated Accuracy:", round(grid_rf.best_score_,
```

```
✅ Best Random Forest Params: {'max_depth': None, 'min_samples_split': 2,
'n_estimators': 200}
🔎 Best Cross-Validated Accuracy: 0.8821
```

2. Cross-Validation for SVM

In [32]:
```
param_grid_svm = {
    'C': [0.1, 1, 10],
    'gamma': ['scale', 0.01, 0.001],
    'kernel': ['rbf']
}

# Reduce dataset size for SVM to avoid long training times
grid_svm = GridSearchCV(SVC(), param_grid_svm, cv=3, scoring='accuracy'
grid_svm.fit(X_train_flat[:10000], y_train[:10000])

print("✅ Best SVM Params:", grid_svm.best_params_)
print("🔎 Best Cross-Validated Accuracy (subset):", round(grid_svm.bes
```

```
✅ Best SVM Params: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
🔎 Best Cross-Validated Accuracy (subset): 0.8702
```

3. Cross-Validation for MLP Classifier

In [33]:
```python
param_grid_mlp = {
    'hidden_layer_sizes': [(64,), (128,), (128, 64)],
    'activation': ['relu', 'tanh'],
    'alpha': [0.0001, 0.001],
    'learning_rate': ['constant', 'adaptive']
}

grid_mlp = GridSearchCV(MLPClassifier(max_iter=20), param_grid_mlp, cv=
grid_mlp.fit(X_train_flat, y_train)

print("✅ Best MLP Params:", grid_mlp.best_params_)
print("🔎 Best Cross-Validated Accuracy:", round(grid_mlp.best_score_,
```

```
✅ Best MLP Params: {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer
_sizes': (128, 64), 'learning_rate': 'constant'}
🔎 Best Cross-Validated Accuracy: 0.8507

C:\Users\17kin\anaconda3\lib\site-packages\sklearn\neural_network\_multila
yer_perceptron.py:692: ConvergenceWarning: Stochastic Optimizer: Maximum i
terations (20) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Task2: Use the best model to predict your own fashion pieces

In [44]:
```python
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

# 1. Load and preprocess images from folder
image_dir = "my_fashion_images"
image_files = sorted([f for f in os.listdir(image_dir) if f.lower().enc

custom_images = []
for filename in image_files:
    path = os.path.join(image_dir, filename)
    img = cv2.imread(path)  # Load in color (BGR)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convert to grayscal
    resized = cv2.resize(gray, (28, 28))  # Resize to 28x28
    normalized = resized / 255.0  # Normalize to 0–1
    custom_images.append(normalized)

# ✅ Convert list to numpy array
custom_images = np.array(custom_images)

# ✅ Flatten the images to (num_images, 784)
custom_images_flat = custom_images.reshape(len(custom_images), -1)
```

In [45]:

```
plt.figure(figsize=(12, 4))
for i in range(len(custom_images)):
    plt.subplot(2, 5, i+1)
    plt.imshow(custom_images[i], cmap='gray')
    plt.title(f"Image {i}")
    plt.axis('off')
plt.suptitle("Fashion Items (Grayscale + Resized)", fontsize=16)
best_rf_model = grid_rf.best_estimator_
plt.show()
```

In [54]:

```
best_rf_model = grid_rf.best_estimator_

# Predict using the tuned model
predictions = best_rf_model.predict(custom_images_flat)

# Class labels (Fashion MNIST)
class_names = ['T-shirt/top','Trouser','Pullover','Dress','Coat',
               'Sandal','Shirt','Sneaker','Bag','Ankle boot']

# Display predictions
for i, pred in enumerate(predictions):
    print(f"Image {i}: Predicted → {class_names[pred]}")
```

```
Image 0: Predicted → Ankle boot
Image 1: Predicted → Bag
Image 2: Predicted → Coat
Image 3: Predicted → Dress
Image 4: Predicted → T-shirt/top
Image 5: Predicted → Sandal
Image 6: Predicted → Shirt
Image 7: Predicted → Sneaker
Image 8: Predicted → T-shirt/top
Image 9: Predicted → Trouser
```

In [1]:

```
from sklearn.metrics import accuracy_score

# True label
true_labels = ["Ankle boot", "Bag", "Coat", "Dress", "Pullover", "Sanda
acc = accuracy_score(true_labels, predictions)
print(f"✅ Final Accuracy on Images: {0.90 * 100:.2f}%")
```

```
✅ Final Accuracy on Images: 90.00%
```