```python
#%% TASK 1

#%% Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Last name Nordlund => A=1

#%%

#%% Basic rules
# 0 right, 1 down, 2 left, 3 up
sp = np.array([
    [1,3,-1,-1], [2,4,0,-1], [-1,5,1,-1],
    [4,6,-1,0], [5,7,3,1], [-1,8,4,2],
    [7,-1,-1,3], [8,-1,7,5], [-1,-1,7,5]
    ]) # defines what state results from a step in each direction
    # depending on start state

number_of_states = sp.shape[0] # rows
number_of_actions = sp.shape[1] # columns

def step(s, action):
    next_state = sp[s,action]
    reward =- 1

    # from rightmost states going right (states 2, 5, 8 action 0)
    if (s==2 and action==0) or (s==5 and action==0) or (s==8 and action==0):
        reward=-1+1

    # from leftmost states going left (states 0, 3, 6 action 2)
    if (s==0 and action==2) or (s==3 and action==2) or (s==6 and action==2):
        reward=-1-1

    # from bottom states going down (states 6, 7, 8 action 1)
    if (s==6 and action==1) or (s==7 and action==1) or (s==8 and action==1):
        reward=-1+1

    # from top states going up (states 0, 1, 2 action 3)
    if (s==0 and action==3) or (s==1 and action==3) or (s==2 and action==3):
        reward=-1+2

    return next_state, reward

#%% Q-table calculated by hand
Manual_Q = np.array([ # actions: right, down, left, up in order
    [0,-1,-2,1], [0,-1,0,1], [0,-1,0,1],
    [-1,-1,-2,0], [-1,-1,-1,0], [0,-1,-1,0],
    [-1,0,-2,-1], [-1,0,-1,-1], [0,0,-1,-1]
    ])
Manual_Q_df = pd.DataFrame(data=Manual_Q,columns=['go right', 'go down',
                                                  'go left', 'go up'])

#%% Init Q-table
Q_values=np.full((9,4), 0.0)
Q_values_df=pd.DataFrame(data=Q_values,columns=['go right','go down',
                                                'go left','go up'])

#%%

#%% Training in two steps
```

```
Q_values_temp = Q_values.copy()
for i in range(2):
    Q_values_temp=Q_values.copy()
    for s in range(number_of_states):
        for a in range(number_of_actions):
           next_state,reward=step(s,a)

           if next_state==-1:
               Q_values_temp[s,a]=reward
           else:
               Q_values_temp[s,a]=reward+np.max(Q_values[next_state])

    Q_values_2step=Q_values_temp.copy()
Q_2step_df = pd.DataFrame(data=Q_values_2step,columns=['go right','go down',
                                                       'go left','go up'])
# Has a couple of errors compared to the manually calculated table

#%%

#%% Training block
for i in range(100):
    Q_values_temp=Q_values.copy()
    for s in range(number_of_states):
        for a in range(number_of_actions):
           next_state,reward=step(s,a)

           if next_state==-1:
               Q_values_temp[s,a]=reward
           else:
               Q_values_temp[s,a]=reward+np.max(Q_values[next_state])

    Q_values=Q_values_temp.copy()

Q_values_df=pd.DataFrame(data=Q_values,columns=['go right','go down',
                                                'go left','go up'])
# This table perfectly aligns with the one calculated by hand, woohoo

#%% TASK 2

#%% Imports
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn import metrics

# Last name Nordlund => A=1

#%%

#%% DataFrame
Df_fashion_test = pd.read_csv('C:/Users/Victor Nordlund/Documents/Github/fashion-mnist_test.csv'
Df_fashion_train = pd.read_csv('C:/Users/Victor Nordlund/Documents/Github/fashion-mnist_train.cs
# concatenating all the objects since I'm doing my own 80/20 splits later
Df_fashion = pd.concat([Df_fashion_test, Df_fashion_train])
```

```
# select my labels, A = 1
my_fashion = (Df_fashion.label == 5) | (Df_fashion.label == 6) | (Df_fashion.label == 7) | (Df_f
my_fashion_df = Df_fashion[my_fashion]
my_fashion_features = my_fashion_df.drop('label', axis=1)

print(my_fashion_df['label'].value_counts())    # Perfectly balanced,
                                                # as all things should be

#%% Plot a couple of images
for i in range(5):
    img = my_fashion_df.iloc[i, 1:].to_numpy().reshape(28, 28)  # ignores label
    plt.subplot(1, 5, i + 1)
    plt.imshow(img, cmap='gray')
    plt.axis('off')

plt.show()

#%%
'''
Types of items in MNIST Fashion:
5          Sandal
6          Shirt
7          Sneaker
8          Bag
9          Ankle boot
'''

#%% 2D PCA, no normalization
# code below roughly taken from the PCA normalization file and modified to
# fit this task
pca_model = PCA(n_components=2)
principalComponents = pca_model.fit_transform(my_fashion_features)

fig,ax = plt.subplots(figsize=(5, 5))
scatter=ax.scatter(principalComponents[:,0],principalComponents[:,1],
                   c=my_fashion_df.label,cmap='Set1')
ax.set_title('2D PCA on Fashion')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

unique_labels = my_fashion_df['label'].unique()
handles = scatter.legend_elements()[0]
labels = unique_labels.tolist()
plt.legend(handles, labels, title="Labels")
'''
From the looks of this plot, Shirts should be very
easy to classify, while there
are large overlaps between especially Bags and
Sandals (which doesn't make much sense to me),
but also Sandals and Ankle boots (which makes
more sense). There also appears to be a lot of
overlap between Ankle boots, Sandals and Sneakers,
which makes sense since they're all examples of
footwear
'''

#%%

#%% 2D PCA, normalized
scaler = StandardScaler()
features_scaled=pd.DataFrame(scaler.fit_transform(my_fashion_features))
```

```
# Scales

pca_model = PCA(n_components=2)
principalComponents = pca_model.fit_transform(features_scaled)

fig,ax = plt.subplots(figsize=(5, 5))
scatter=ax.scatter(principalComponents[:,0],principalComponents[:,1],
                   c=my_fashion_df.label, cmap='Set1')
ax.set_title('2D Normalized PCA on Fashion')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

unique_labels = my_fashion_df['label'].unique()
handles = scatter.legend_elements()[0]
labels = unique_labels.tolist()
plt.legend(handles, labels, title="Labels")
'''
Mostly aligns with the previous plot. The main difference
is that the Sneaker points aremore scattered, especially
towards the top right corner of the plot. The Ankle boots
points alsoappear slightly easier to classify than previously.
'''

#%%

#%% Making X and y arrays of the DataFrame
X = np.asarray(my_fashion_features) # features
y = np.asarray(my_fashion_df['label']) # target


#%% Random forest classification
clf = RandomForestClassifier()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=None,
                                                    stratify=y)
# since there is an equal amount of items for each type,
# both training and test data are balanced

#%% Running RFC

clf.fit(X_train, y_train)
test_prediction=clf.predict(X_test)
confusion_matrix = metrics.confusion_matrix(y_test,test_prediction)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = [5, 6, 7, 8, 9])
cm_display.plot()
accuracy_RF=np.trace(confusion_matrix)/np.sum(confusion_matrix)
# 96.3%, most of the errors are from sneakers and ankle boots

#%% Feature importance extraction
feature_importance = clf.feature_importances_
importance_df = pd.DataFrame({'Feature': my_fashion_features.columns,
                              'Importance': feature_importance})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

print(importance_df.head(3)) # returns pixels 93, 234 and 378

#%%
pixel_93_row = 93 / 28 # ~ 3.32
pixel_234_row = 234 / 28 # ~ 8.36
```

```
pixel_378_row = 378 / 28 # = 13.5
'''
Pixel 93 should be on the 4th row, roughly one third of the way in from the
left
Pixel 234 should be on the 9th row, roughly one third of the way in from the
left
Pixel 378 should be right in the middle of the 14th row, right in the middle
of the 28*28 square
From my limited scrolling through of the dataset, it seems like almost all of
the edge pixels don't contain any information and should thus have low
importance. It makes sense that at least one of the pixels in the middle (378)
is of high importance. However it's not the one of highest importance, both 93
and 234 are more towards the top left corner and from what I can tell, the
footwear items don't seem to have any information there, which makes it an
effective quadrant to check for classification.
'''
```