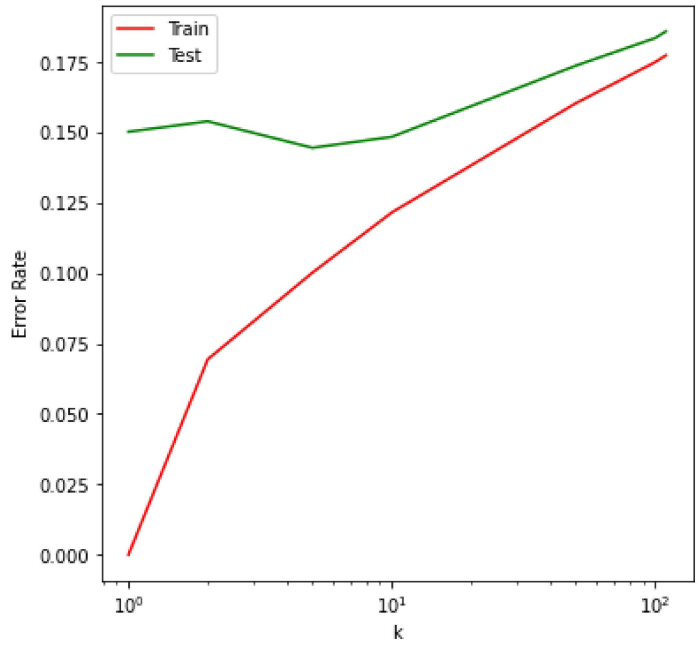
In [19]:
```
import torch
import torchvision
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
import numpy as np
import matplotlib.pyplot as plt



batch = 128 # batch size (change to ensure GPU occupay)
coded_channels = 8 # channels of the output of encoder
learning_rate = 1e-3
dorpout_p = 0.5

# load weight from .pth file
pretrained = False

# get the device to run
device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
print(f'Selected device: {device}')

training_data = torchvision.datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=torchvision.transforms.ToTensor()
)

training_y = training_data.targets
training_x = training_data.data.numpy().reshape(-1, 28*28)


test_data = torchvision.datasets.FashionMNIST(
    root="Fashion_MNIST_data",
    train=False,
    download=True,
    transform=torchvision.transforms.ToTensor()
)
test_y = test_data.targets
test_x = test_data.data.numpy().reshape(-1, 28*28)
```

```
Selected device: cuda
```

Error rate with KNN's complexity

In [22]:
```
figure, axes = plt.subplots(1, figsize=(6, 6))
num_neighbors =[1, 2, 5, 10, 50, 100, 110]

err_tr = np.zeros(len(num_neighbors))
err_te = np.zeros(len(num_neighbors))

for i,k in enumerate(num_neighbors):
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(training_x,training_y)


    err_tr[i] = 1. - knn.score(training_x, training_y)
    err_te[i] = 1. - knn.score(test_x, test_y)

axes.semilogx(num_neighbors, err_tr, c='red', label='Train')
axes.semilogx(num_neighbors, err_te, c='green', label='Test')

axes.set_xlabel('k')
axes.set_ylabel('Error Rate')

axes.legend()
```

Out[22]:
```
<matplotlib.legend.Legend at 0x2b0d105e040>
```

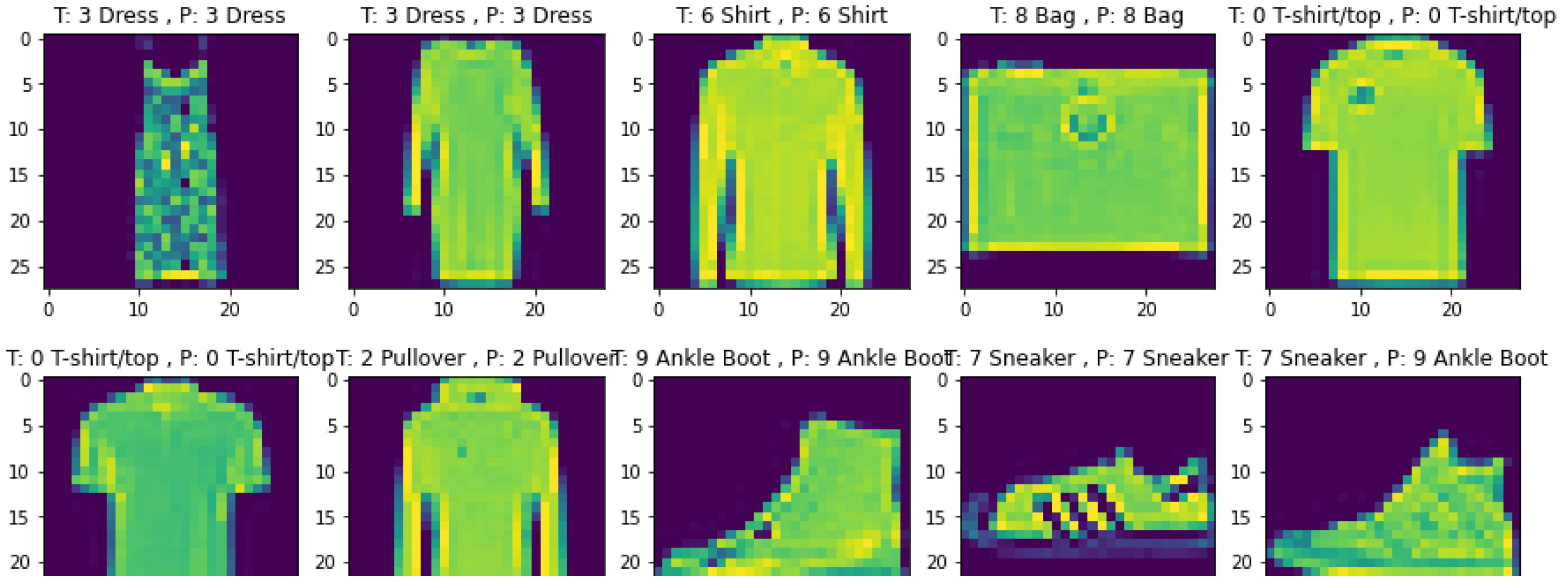


Training time & Prediction time

In [53]:
```
import time


knn_5 = KNeighborsClassifier()
start_fit = time.time()
knn.fit(training_x,training_y)
end_fit = time.time()

start_predict = time.time()
precict = knn.predict(test_x)
end_predict = time.time()

time_for_fit =end_fit-start_fit
time_for_predict = end_predict-start_predict

print(f"Time for fit the whole data: {time_for_fit}")
print(f"Time for making prediction of the whole data: {time_for_predict}")
```

```
Time for fit the whole data: 0.003997087478637695
Time for making prediction of the whole data: 14.292734384536743
```

In [54]:
```
# do some visualization
classes = ('T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle Boot')
dataset = test_data
figure, axes = plt.subplots(10, 5 ,figsize=(15, 35))


for i in range(50):
    index = i+300
    img = test_x[index].reshape(28, 28)
    label = classes[test_y[index]]
    predict = classes[precict[index]]
    axes[i//5,i%5].imshow(img)
    axes[i//5, i%5].set_title(f'T: {test_y[index]} {label} , P: {precict[index]} {predict}')


plt.show()
```



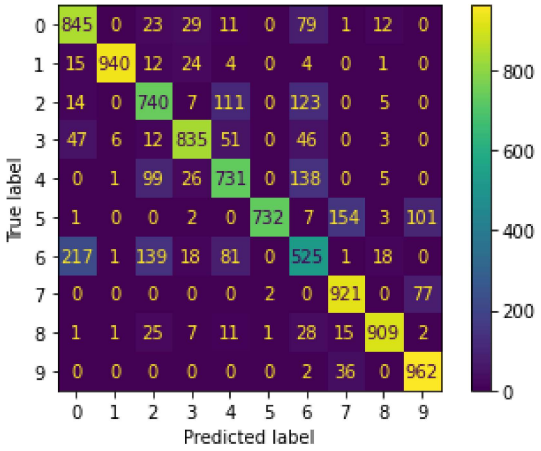
Type *Markdown* and LaTeX: α^2

In [ ]:
```

```

Confusion Matrix

In [46]:
```
cm = confusion_matrix(test_y,precict)
cm_disp = ConfusionMatrixDisplay(confusion_matrix = cm)
cm_disp.plot()
```

Out[46]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2b0dcc20580>



In [ ]:
```
```